DOCUMENT RESUME

ED 050 593	EM 008 953
AUTHOR	Zinn, Karl
TITLE	Requirements for Programming Languages in Computer-Based Instructional Systems.
INSTITUTION	Organisation for Economic Cooperation and Development, Paris (France). Centre for Educational Research and Innovation.
REPORT NO	TR-CERI-CT-70.62
PUB DATE	5 Mar 70
NOTE	78p.; Paper presented at the Conference on the Use of Computers in Higher Education (Paris, France, March 19-21, 1970)
EDRS PRICE	EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS	*Computer Assisted Instruction, Evaluation Criteria, Interaction, Intermode Differences, *Performance Criteria, *Performance Specifications, Program Costs, *Programing Languages

ABSTRACT

The author reviews the instructional programing languages which already exist and describes their methods of presentation, organization, and preparation. He recommends that all research and development projects remain flexible in their choice of programing language for a time yet. He suggests ways to adapt to specific uses and users, to exploit the interactive mode of computer use, and to coordinate language maintenance and system operations with project goals. Drawing on a variety of sources, he lists criteria for examining a language in relation to a given user or goal. He discusses some general effects a language may have on instructional strategies and notes that a computer's record-keeping ability is a great asset in assessing instructional strategies. He feels serious consideration must be given to a means for translating instructional materials and strategies from one institution and system to another. He delineates the limitations placed on a language by its hardware, software, and communication system and warns that time-sharing systems may need larger development resources than instructional researchers can afford. He cites desirable features such as adaptability, economical operation, and documentation which should be considered in designing an instructional system. A glossary is provided. (JY)

ED050593

centre
for
educational
research
and
innovation

REQUIREMENTS FOR PROGRAMMING LANGUAGES IN COMPUTER-BASED INSTRUCTIONAL SYSTEMS

technical report

EM 008 753

OECD

ORGANISATION FOR ECONOMIC CO-OPERATION AND DEVELOPMENT

ED050593

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

ORGANISATION FOR ECONOMIC
CO-OPERATION AND DEVELOPMENT

Centre for Educational Research
and Innovation

CERI/CT/70.62

CONFERENCE ON THE USE OF COMPUTERS IN HIGHER EDUCATION

held in Paris from March, 19th-21st, 1970
(Joint Project CERI XI)

Requirements for programming languages in Computer-based instructional systems

by Mr. K. ZINN

Center for Research
on Learning and Teaching
University of Michigan
Ann Arbor, USA

79522

CONTENTS

page

PREFACE 7

I. INTRODUCTION

1. Purposes 13
2. Audience 14
3. Sources of other interpretation and opinion . . . 14

II. KINDS OF INSTRUCTIONAL PROGRAMMING

1. Organising the content and preparing the procedures 17
2. Description of successive frames or items 19
3. Provision for conversation within a limited context 24
4. Description of a standard procedure by which material is presented 25
5. Specification of an environment for programming and problem solving 29
6. Relative use of the four kinds of programming . . 33

III. RECOMMENDATIONS

1. Maintain flexibility 37
2. Adapt to specific uses and users 38
3. Exploit interactive mode of computer use 39
4. Relate language maintenance and system operation to project goals 39

IV. STATEMENTS OF POSITION OR POINTS OF VIEW*

1. ASSESSMENT OF A LANGUAGE AND SYSTEM

A. Representations for Language and System Characteristics

1. Functional aspects compared against a standard list 41

---

* These statements were selected from published and unpublished literature (sources are given at the end of the section) and paraphrased to convey the intent in a few lines within the context of the larger outline and to point out similarities and differences among related positions.

Because many of the statements were taken from a comparative study of programming languages, already a distillation of points of view, the list of references does not include all primary sources.

2. Samples of code, presumably typical of user and task requirements 42
3. Empirical measures of usefulness: programming time, errors, attitude, execution time, etc. 42

B. Suitability for the Primary Users, Considering Background and Goals

1. Student 43
2. Instructor (supervisor or manager of learning) 44
3. Counsellor or administrator, if different from the instructor 44
4. Author or lesson designer 44
5. Researcher on instruction and learning . . 45
6. Researcher on the systems and information sciences 45

C. Suitability for the Mode of Computer Use

1. Routinised drill and testing 46
2. Computerised programmed instruction 46
3. Diagnosis and remediation 46
4. Question answering 47
5. File and text manipulation 47
6. Numerical problem solving 47
7. Graphic display 48
8. Other modes not defined here (i.e. growth potential to meet unanticipated user needs). 48

D. Suitability for the Style of Program Preparation

1. Description of successive frames or items 49
2. Provision for conversation within a limited context 50
3. Description of a standard procedure by which material is presented 50
4. Specification of an environment for programming and problem solving 51
E. Implementations Available (e.g. Which Machines and What Memory Size) 51
F. Documentation, Teaching Aids and System Maintenance Available 52

2. GENERAL IMPLICATIONS OF A LANGUAGE AND ITS IMPLEMENTATION FOR STRATEGY OF INSTRUCTION

A. Data Available for Automatic Decisions, e.g. Student Response Time, Records from Previous Day, and Group Averages 53
B. Processing Capability, Handling Character as well as Numeric Information . . . . * 53
C. Adaptability to Specific Tasks, e.g. Convenience for Describing Models, Drawing Diagrams or Retrieving Information . . . . * 53
D. Generality of Procedures, e.g. Separation of Procedure from Content, and Generation of Material from General Rules 53
E. Manipulation of Files, e.g. Directories, Curriculum Material, and Performance Data . . . 54

3. GENERAL CONSIDERATIONS OF UNIVERSALITY

A. Universal Language by Established Standard . . . 55
B. A Few Common Languages as Justified by Different Requirements 55
C. Automatic and "Manual" Translation Among Languages of Similar Purpose 56
D. Communication and Documentation with a "Publication" Language 57
    1. Among curriculum developers 58
    2. To reviewers and potential users 58
    3. To programmers 59
E. Natural versus Formal Language 59

4. SYSTEM LIMITATIONS ON LANGUAGE (cf.II on implications for strategy)

A. Hardware, e.g. Clock, Interrupt, Workspace in Memory, and Terminals 60
B. Software, e.g. Files, Editing, Linkage, Concurrent Users 60
C. Communications, e.g. Transmission Bandwidth, Time and Cost. 61
D. Summary of Cost Considerations 62

* These sections are not included in this draft. They will be presented in a later publication.

5. DESIGN CONSIDERATIONS

A. Adaptability 63
B. Economics 63
C. Modularity 64
D. Documentation 64

6. INTERACTIVE MODE CONTRIBUTIONS TO LEARNER AND AUTHOR

A. Immediate and Responsive Reply 65
B. Ease of Conducting a Dialogue and Learning the Rules 65
C. Flexibility During Working Session 66

V. REFERENCE WITH ANNOTATIONS 67

VI. GLOSSARY FOR COMPUTER USES IN EDUCATION 75

PREFACE

The following paper, prepared by Professor Karl ZINN of the Center for Research on Learning and Teaching, University of Michigan, Ann Arbor, United States, covers a very specific aspect of the use of computers as an instrument of teaching.

While a brief introduction to the content of this paper will undoubtedly be useful to readers, the main need would appear to indicate the reasons why the Centre for Educational Research and Innovation (C.E.R.I.) of the OECD has undertaken a multinational "concerted drive" in such a promising as well as controversial field. The following are some which should be noted:

(1) The pressure of requirements in the matter of education, especially in higher but also in secondary education, which makes it important and even essential to achieve a better allocation of resources. Some "moments" in the educational and learning process can thus be facilitated by the use of teaching aids, though of course under certain conditions. As a result, from this point of view alone the teacher, and, better still, the teacher team, are freer to meet the specific needs of students where direct contact with them is necessary. Audio-visual aids are thus part of such an equipment array, and meet with the degree of success and the difficulties we know. From this same point of view but at a quite different level as to the functions it can assume in teaching processes, the computer has already found its way into education, if only better to explore its possibilities - and the limits to its utilisation - and avoid any such inordinate use as already reported in some cases, the question had to be approached from the angle of Research and Development. In so doing, it should be emphasized that the Centre for Educational Research and Innovation is simply filling its role as catalyst and "think tank" on behalf of the OECD countries, whether in this field or in others which together make up its programme of work. The objective, especially here, is quite clear - to produce a certain number of recommendations which can help national authorities to define their policies.

(2) The need for better understanding of the teaching and learning processes. Parallel to its use as an instrument of education, the computer can already be regarded as one of the most effective tools of purposefully experimental pedagogical research. In the first place - provided, of course, that it is "equipped" with adequate programmes - it is an incomparable analytical instrument owing to its close accuracy and the opportunity it offers for dialogue, and in the second place it is capable of so managing specific teaching situations that optimal results can be obtained both by students and teachers.

The present trend towards the individualisation of education and the introduction of permanent knowledge testing indicate that it will very soon be necessary to use computers. There are many possible fields of application: analysis of curricula (identification of difficulties in regard to the conceptual organisation of content, forms of data presentation and individual pupil characteristics), the planning of steps adapted to the specific objectives pursued by the students, etc.

(3) Existence of a "technology of education": The term is used in a restricted sense - one which obliges both teachers and the educational authorities to specify the ways and means of using the computer without running the risk of producing some kind of caricature of the educational process - "the education machine" for example. Furthermore, as strictly technological progress becomes more rapid, it follows that the cost of equipment will decrease. This being so, if sufficient serious thought is not promptly given to the opportunities for using computers in education the danger is that these will have been installed in educational establishments before the human and structural aspects and the content of education has had a chance to be re-assessed in the light of these new factors. Clearly this cannot be allowed to happen.

These are not the only considerations which have induced C.E.R.I. to undertake its concerted drive, one by-product being the following paper.

This concerted drive, as co-ordinated by C.E.R.I., calls for the direct participation of five universities in OECD Member countries.(1)

---

(1) France: Paris University (VII) - Faculty of Science - "Computer for students" Laboratory - Dir.: Professor Le Corre and Professor R. Jacoud.

Belgium: Louvain University - General Physics Laboratory - Dir.: Professor A. Jones.

Netherlands: Leiden University - Department of Education - Dir.: Professor L. de Klerk.

Japan: Osaka University - Faculty of Arts - Dir.: Professor S. Tanaka.

Great Britain: Cambridge University - Department of Applied Mathematics and Theoretical Physics - Dir.: Professor G.K. Batchelor.

To this first "network" should be added a certain number of other higher-education establishments in Europe ("Technische Hoschule", Aachen, Germany - C.S.A.T.A., Bari University, Italy - "Computer Based Learning Projects", Department of Education, University of Leeds, Great Britain, etc. Contacts are also maintained with American universities through the U.S. Office of Education. This "joint project" (so named because of the form of financial contributions and responsibilities of the parties concerned) was launched in 1969 with the essential object of providing, first, recommendations to national authorities (see paragraph ii), and secondly some of the answers to the more technical questions which must be solved if computer-assisted education is not to develop haphazardly or in response to certain pressures outside or inside education but rather in order to help improve the educational process.

These questions include:

(i) Problems connected with the importance from the pedagogic point of view of methods of using computers as an instrument of education, especially:

- How can computers be introduced into the structure of education? What are the most suitable strategies for introducing them?
- What are the possibilities of combined use of computers and other teaching methods, especially audio-visual methods and television?
- How can participation by the student be made more active?
- What is the influence of the computer on the behaviour of students and teachers? How can the former be trained to use the instrument so that the maximum information may be obtained from it?
- In what cases should the computer, and programming procedures for teaching in general, be abandoned?
- What part should be played by evaluation?

(ii) Problems concerning the drafting of courses, questionnaires etc.:

- What is the best way of numbering items and questions?
- How frequently, and where should there be items of recapitulation and overall presentation?
- Are different structural arrangements desirable depending on the subject being taught?
- How should the various media (manual, films, photographs, simple or more complex terminal, etc.) be allocated?

(iii) Problems of equipment:

- Should one use computers especially adapted for education, or all-purpose computers?
- Should preference be given to large computers (central store more than 9 K) with a large number of terminals or to smaller computers (of the order of 4 K) with a limited number of peripheral terminals?
- Is it possible to use older (second generation) computers?
- Should simple or sophisticated terminals be selected?
- Should transmission be slow (below 100 tapes) or rapid?
- How should the interface be designed?

(iv) Language problems:

- Communication between pupil and machine
- How should the theoretical problems raised by free answers be solved?
- Can modular blocks be used for analysing answers (by enumeration, range of numbers, arithmetical answers, etc.)?
- How can multi-lingual adaptation of the course subject matter be achieved?
- Communication between teacher and machine:
- How can languages for simulation suitable for case studies be developed?
- Can standard languages for detecting errors be developed for course preparation purposes?

(v) Problems of cost regarding use of the computer:

- On what basis and by reference to what should the cost be calculated?
- What is the return on the system compared with other methods of teaching?

Independently of the work carried out in this group, it was agreed, again for the sake of international co-operation, periodically to organise wider meetings so that a larger number of experts on the subject could discuss items of the programme of work for this concerted drive. The agenda of one of these meetings, held at the OECD in March 1970, provided for discussion of the subject "Languages in computer-based instructional systems". It was on this occasion, in order to open the discussion as well as present a background material regarding work in this field, that C.E.R.I. asked Professor K. Zinn to prepare the following paper.

It is the first of a series which will include several titles. At first sight it may seem surprising that the series should begin by a technical report apparently designed for specialists alone, however numerous they may be. In fact, the problem in question illustrates the confusion and contradictions that frequently exist in the matter of technology of education. One might even say that it is a typical example of a problem which is both true and false:

- false in that in theory as well as practice it is impossible to develop an all-purpose language able to deal anymore than adequately with all teaching situations. Any persistent attempt to develop such a language would lead to the refinement of stereotyped "dialogues" which would not allow for the sometimes baffling aspects of any teaching and learning process. We for our part are convinced that there are other more urgent questions to be dealt with.

- true in that where there is dialogue there must be language! However sophisticated this may be it is nevertheless subject to certain constraints, and what is true for natural languages with all the scope they offer for ambiguity and implication ("entropy" as it were) is still more true for artificial languages, however comprehensive they may be (FORTRAN, ALGOL, etc.).The computer cannot in fact admit of the slightest error and in case of doubt it can only forbear and send back the question. In these circumstances it is important that the teacher who intends to programme a certain part of his course on the computer should be able to do so with the maximum of security and accuracy. It is also essential - which is where the complications set in - that the language used should be sufficiently flexible to enable it to cover the diversity of teaching situations by taking both teaching requirements and students' reactions into account, and also be easy to master if the teacher is not gradually to turn into a computer scientist and neglect his duties as a teacher.

The principal merit of this study by Professor K. Zinn is, first, that it reviews what already exists and lists the many languages so far developed according to a certain number of more general criteria. At the same time the suggestions and recommendations it contains provide a basis for discussion which, it may be hoped, will yield positive results both from the standpoint of the joint project co-ordinated by the C.E.R.I. and that of all those who desire that an appropriate use be made of computers in education.

I. INTRODUCTION

1. Purposes

The primary goal of this paper is to put forth for criticism and further discussion some tentative recommendations on instructional programming languages. I have tried to make my statements specific enough to be criticised in detail and perhaps to be improved through elaboration, yet general enough that they might be applied in discussion of various computer-based systems developed for differing purposes. Certainly I shall not achieve sufficient detail and the desired generality in this draft. Although my recommendations are derived from considerable experience with a variety of programming systems and curriculum authors, they should be interpreted with caution in their present tentative state. Guided by comment from readers, I hope to progress to a more definite and useful statement in the near future.

In order that the recommendations can more readily be criticised by the reader, I have included the skeleton of information from which they were derived. The format used for this background information is an outline of language and system considerations on which various positions or points of view have been expressed. Numerous aspects of programming languages and user support are characterised by two or more persons, with some overall interpretation spread throughout as guidance for the reader.

In order to provide convenient access to primary sources, writings with direct application to the problems discussed are listed with brief annotations. Since information resources for instructional use of computers continue to change rapidly, I have included some suggestions for obtaining current information and viewpoints as they are written and discussed.

A secondary goal for this paper, perhaps important to some readers, is to provide in the early pages a short introduction to the role of programming languages in the preparation and use of computer-based learning exercises. Various types of programming are exemplified with brief discussion of implications for the style and goals of instruction accomplished. Some knowledge of computers and instructional uses is assumed; for an introduction to the field and a guide to general sources, write the ERIC Clearinghouse on Educational Media and Technology, Cypress Hall, Stanford University, Stanford, California 94305.

2. Audience

I should like to reach potential users of computer-based systems in education, especially those supervisors and administrators who are now making decisions about economical introduction of effective computer aids into instructional programs for which they are responsible. Although I intend the discussion in this paper to be useful to a reader not already versed in computer use, the present draft surely needs much more work, and the substance is likely to be revised on the basis of discussion among experts. Perhaps, then, just those experts should be considered a primary (and critical) audience for this draft.

3. Sources of other interpretation and opinion

Although I have attempted to incorporate opinion of others in this paper on programming languages, representation of their ideas can be only incomplete and biased at best. A student of programming languages should look at a number of other sources. The ERIC guide to information already has been mentioned as an introduction and general source. A few specific items are mentioned here for the reader who needs no introduction to the topic.

E. N. Adams* provided a comprehensive tutorial presentation on "Technical considerations in the design of an instructional system" for the NCET (UK) Symposium on Computers in Education held in Leeds in 1969. His consideration of languages and instructional programming is very relevant to the issues discussed in this paper.

The perspective of C. Victor Bunderson, head of the CAI Laboratory at the University of Texas at Austin, can be inferred from the projection of curriculum development and use on computer-based systems which he presented at the American Educational Research Association meeting in February of 1969. Bunderson has provided more specific recommendations in notes for a three-day seminar on instructional use of computers, but these are as yet unpublished.

Charles Frye of System Development Corporation, Santa Monica, California, provided a discussion of languages in 1968 using four categories: conventional compiler languages, adapted compiler languages, interactive computing languages, and specially devised instructional author-languages. Some languages were assigned to the wrong category, but the concepts expressed are useful.

---

* Complete references to publications are given in the bibliography.

Educom in Boston distributes the final report of a comparative study pursued in 1967-68 with support from the U.S. Office of Naval Research and the University of Michigan (as well as the Kellogg Foundation, which supports Educom administrative and pilot efforts) Information on over 40 programming languages is included. Although I was the primary author of that report, the opinions of contributors come through more clearly in that document than in the present summary paper. Because technical information goes out of date very rapidly, watch for a revision or replacement for the Educom document before the end of 1970.

J. Donio of OECD-IRIA described the "present situation and current trends" after his visit to the USA in 1968, including attention to problems and proposals regarding languages.

William Ramage edited the presentations and transcript of discussion by a dozen specialists gathered at the University of Pittsburgh in 1967 to discuss CAI author languages. Although many of the comments are now outdated, most of the requirements set down at that time have not yet been met.

The proceedings of most conferences, working sessions and national commissions dealing with instructional use of computers include some consideration of language requirements and implication for strategy and economics. For example, the report of the Commission on Instructional Technology (USA) is soon to be released and the advance proceedings of an August conference on computers in the teaching of physics and mathematics will include substantial sections on different kinds of languages.

These and other sources of information and opinion are annotated in the list of references included near the end of this document.

II. KINDS OF INSTRUCTIONAL PROGRAMMING*

The current state of instructional programming languages is characterised by proliferation of independent efforts to produce an all-purpose language and by unstated assumptions about appropriate uses of computers for instruction. Some of the new languages, having been motivated by deficiences in old ones, turn out to be only superficially different. A comparative study of programming languages listed over 40 different languages and dialects which have been developed especially for instructional use of computers, and the differences among them are not very great in most comparisons. Attention to the style of programming typically done with different languages can simplify a topic which has been made unnecessarily complicated by obstructions to communication among those developing the programming languages and others devising instructional strategies.

1. Organising the content and preparing the procedures

Some types of instructional programming for computers are likely to be used by subject experts (curriculum writers) and other styles favour computer specialists. Typically computer and discipline experts must collaborate if the result is to exploit the computer contribution as well as relate to real instructional problems and appropriate organisations of the subject matter. The distinction between substance and procedure is blurred because most programming languages have combined the two aspects and initial successes depended on unusual individuals skilled in both areas.

The development of computer-based curriculum can be separated into two or more processes for my purpose of distinguishing content and procedure; I would not apply this analysis to all computer uses. The most obvious process is arranging the substance or content of instruction in a way which is appropriate for computer presentation and for interaction between learner and the computer-based representation of the knowledge base. No less important is the creation of a set of procedures by which the curriculum is delivered and the knowledge base is explored by each individual student. Secondly, the curriculum development group needs a process by which the substance is put into formatted information files on which the procedure programs can operate economically.

---

* The overview and interpretations of this section have been adapted from two previous presentations (Zinn, 1969b, 1970). Detailed suggestions from E.N. Adams, Fred Bennik and Charles Frye helped considerably in its revision.

The first process, organising the content, does not require a programming language at all, although the designer is likely to work within specific formats and within system constraints interpreted for him by someone expert in programming. Although demonstrations and exploratory work have been prepared directly in programming languages, some of them called "author" languages, curriculum development efforts which are successful on a large scale almost invariably adopt stylised forms and standard procedures to represent learning materials. The so-called author language for a project (for example, Coursewriter II) is used only by programmers who make a specialty of it, and the discipline expert is expected to give full attention to attributes and organisation of the subject matter and learner performance.

The second process, procedure preparation, does not require a specialised language for "author convenience" since the work is done by a programmer, and done only once for large amounts of curriculum presented to many students. Economy in execution is more important than convenience while programming such a procedure. Many projects making instructional use of computers have used programming languages in which instructions require nearly the detail of each machine operation (called "assembly language"), and at times a general-purpose language such as ALGOL or FORTRAN. Some projects which must use an "author language" for execution (for example, Coursewriter II on the IBM 1500 Instructional System) treat it as an assembly language, generating instructions for the Coursewriter processor automatically by the use of some other language which the procedure designer finds more suitable.

The procedure programmer should use a procedure-writing language which includes capability for programming interactive uses, e.g. access to a clock or other timing mechanism, control of automatic interrupts in order to handle difficulties encountered during processing of student constructions, and linkage to files of other users. The system must allow data to be recorded, saved in permanent files from day to day, and recalled when the student wishes to continue where he left off or to check his performance against that of others.

Both processes or tasks require some communication between instructional expert and computer programmer. The essential nature of computers and operating systems does impose some general constraints on what the subject expert may usefully specify, and the objectives of instruction and the means by which they might be achieved have definite implications for the language chosen and the procedures to be written. In summary, I would not expect discipline experts to conceive new techniques for exploiting information processing systems in instruction, nor would I rely on computer experts to determine organisation of and means of access to curriculum.

The third part of curriculum development, getting substance into appropriately arranged files, is a desirable aid to the various team members involved, a time-saver, and, in many cases, a money saver. Coordinated design of this process by curriculum and computer experts is assumed.

For curriculum development projects at least, the language should meet requirements of economical execution for the kind of processing anticipated. Convenience for the programmer is only secondary, since his job is relatively minor and he can work out some way to achieve what is desired.

Convenience for the curriculum designer is essential, but not directly tied to the language of implementation.

I would account for the curious state of languages for programming instructional use of computers today by the fact that virtually all of these languages were prepared by computer programmers who were automating the tedious parts of their job as they viewed it. Perhaps new languages, or the means for producing them, will be derived from a more comprehensive analysis of the requirements of all users involved.

In the following exposition on kinds of instructional programming I use four headings: description of successive frames or items; provision for conversation within a limited context; description of a standard procedure by which material is presented; and specification of an interactive environment for programming and problem solving. The emphasis in this section continues to be on actual programming applications (or use of a language) rather than on apparent capabilities (functional aspects of a language).

2. Description of successive frames or items

The most common application of computers for instruction appears to be an extension of programmed instruction or audio-visual presentation of lectures. It is not surprising that most languages encourage this style of programming: Table 1 gives a tentative assignment of languages for which I have documentation. I do not mean to imply that computerised programmed instruction is all that these languages are capable of representing; I offer that each does encourage that mode of computer use.

These languages may in general be distinguished from scientific and business programming languages by a number of factors: convenience for display of text; acceptance and classification of relatively short strings of text typed by the student (or any user); automatic recording of answers or other performance data; and implicit branching determined by the categorisation of an answer or the contents of a counter which is part of the history of student responses. Although FORTRAN, ALGOL and other languages lacked these features, a new generation of general-purpose programming languages and on-line systems will include convenient facilities for string processing, file access, and definition of normal conventions by which the instructional programming needs are readily accomplished.

IBM's COURSEWRITER is the best known example of a language which encourages the description of frames or items, especially the original version for the IBM 1401. It grew out of a statistics course authored by Ralph Grubb in W.R. Uttal's CAI project using an IBM 650 at Watson Research Center[*] during the early 1960's. Lenore Selfridge was coding instruction materials frame-by-frame according to the logic Grubb defined when she suggested a Teacher Interpretive Program (TIP) to simplify the task of entry and revision of the statistics program. Other authors at Watson Research Center at the time were using other instruction strategies, each programmed individually.

The advantage of using TIP was sufficient to induce other authors to use the same approach - a kind of computerised programmed instruction - and a language called COURSEWRITER achieved status as a general language. Many of the languages in Table 1 were motivated by COURSEWRITER and then developed independently. Most of them have promoted only a frame-oriented conception of computer use. Programming other instruction strategies has been accomplished through additions to the language and special efforts of programming staff other than the curriculum author.

A sample program for a simple mathematics drill exercise is shown in Figure 1. The indentation has been added to indicate some of the implied branching during processing. In general: when a condition is satisfied in a statement the indented statements below it are executed also; when that condition is not met the indented part is skipped. Statement sets preceded by "wa" anticipate certain wrong answers, and record the second occurrence of a certain kind of error. Otherwise, the similarity of the code and

(continued page 24)

---

[*] Uttal, W.R. "On conversational interaction, "*Programmed Learning and Computer-based Instruction*, pp. 171-190, J.E. Coulson (Ed.), New York: Wiley, 1962.

TABLE 1: Description of successive frames or items:

COURSEWRITER I (IBM 1400), II (IBM 1500), and III (IBM 360). Obtain information from IBM Branch Offices.

COURSEWRITER, experimental (IBM 7010). T.J. Watson Research Laboratory, Yorktown Heights, New York 10598.

COURSEWRITER, experimental (IBM 360/50). IBM Systems Development Division, Poughkeepsie, New York 12602.

WRITEACOURSE, Computer Science Group, University of Washington, Seattle, Washington 98105.

LYRIC: Language for Your Remote Instruction by Computer. Computer-Assisted Instruction Systems, 979 Teakwood Rd., Los Angeles, California 90049.

DISCUSS. Information Processing Laboratory, Institute of Library Research, University of California, Berkeley, California 94720.

CAL: Course Author Language. Computing Facility, University of California, Irvine, California 92664.

INFORM. Communications and Electronics Division, Philco-Ford, 3900 Welsh Road, Willow Grove, Pennsylvania 19090.

COMPUTEST, COMPUTEST-II, and PILOT: A computer Language for Individual Testing; and Programmed Inquiry, Learning or Teaching. Computer Center, School of Medicine, University of California, San Francisco, California 94122.

DITCH. Lafayette Clinic, 951 E. Lafayette, Detroit, Michigan 48207.

COPI I and II: Computer-Oriented Programmed Instruction. Educational Systems Programming, Federal Systems Division, UNIVAC Division of Sperry Rand, St. Paul, Minnesota 55116.

DIALOG. Technomics, 1455 19th Street, Santa Monica, California 90404.

MINORCA and GLURP. Center for Educational Software, New England School Development Council, 55 Chapel Street, Newton, Massachusetts 02160.

FOIL: File-Oriented Interpretive Language. Center for Research on Learning and Teaching, University of Michigan, Ann Arbor, Michigan 48104.

MENTOR. Department of Educational Technology, Bolt Beranek and Newman, 50 Moulton Street, Cambridge, Massachusetts 02138.

CAN: Completely Arbitrary Name. Department of Computer Applications, The Ontario Institute for Studies in Education, 102 Bloor Street West, Toronto 5, Ontario, Canada.

TUTOR. Computer-based Education Research Laboratory, University of Illinois, Urbana, Illinois 61803.

PICLS. Purdue Interactive Computer-Aided Learning System, Computer Sciences, Purdue University, Lafayette, Indiana 47907.

TEACH. Department of Physics, University of Arizona, Tucson, Arizona 85721.

CHIMP. Institute for Molecular Physics, University of Maryland, College Park, Maryland 20742.

EXPER, experimental. Information Systems Program, G.E. Research and Development Center, P.O. Box 43, Schenectady, New York 12301.

HAL, experimental. Advanced Development Group, Honeywell Electronic Data Processing Division, 200 Smith Street, Waltham, Massachusetts 02154.

TEACHER/II. Department of Mathematics, University of Denver, Denver, Colorado 80210.

UAL and UIL: UNIVAC Author Language and UNIVAC Interactive Language. Federal Systems Division, Univac Division of Sperry Rand, St. Paul, Minnesota 55116.

PLANIT: Programming Language for Interactive Teaching. Educational Department, System Development Corporation, Santa Monica, California 90406.

```
qu 1. 2^3 = ?
  ca 8
      ty  correct
      ad  1//c2
  wa 6
      ty  No.Did you read it as: 2 x 3?
          Watch for exponents.
      ld  1//s7
  un 2^3 = 2 x 2 x 2
      ad  1//c3
  un  The answer is 8
      ad  1//c4
qu 2. 3^2 = ?
  ca 9
      ty  correct
      ad  1//c2
  wa 8
      ty  No. Did you read it as: 2^3 ?
          Watch the order.
  wa 6
      ty  No. Did you read it as : 3 x 2 ?
          Watch for exponents.
      br  x10//s7//0
      ty  You made this error on the last problem too?
  un 3^2 = 3 x 3
      ad  1//c3
  un The answer is 9
      ad  1//c4
```

Counters:

c2 number correct
c3 number of hints
c4 number of answers

Switches:

s7 if multiplied

Figure 1: From a frame-oriented program using a dialect of Coursewriter

conversation to a programmed text is apparent. In fact, automatic translators have been written so that computer can accept linear (or simple branching) programmed text and derive instructions by which to carry on CAI interaction with a student.

3. Provision for conversation within a limited context

Only a small proportion of computer-based instruction programs of the tutorial variety have been specifically designed to encourage additional initiative on the part of the student, and to provide a relevant reply whatever he may do. The languages in Table 2 have been pulled from the first category because they have one or more additional feature for the purpose: conditional expressions, data recording, text processing, block structure, etc.

TABLE 2: Provision for conversation within a limited context:

MENTOR. Department of Educational Technology, Bolt, Beranek and Newman, 50 Moulton Street, Cambridge, Massachusetts 02138.

ELIZA. Education Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

FOIL and FIT: File-Oriented Interpretive Language, and Flexible Instruction Translator. Center for Research on Learning and Teaching, University of Michigan, Ann Arbor, Michigan 48104.

MINORCA and GLURP. Center for Educational Software, New England School Development Council, 55 Chapel Street, Newton, Massachusetts 02160.

PLANIT: Programming Language for Interactive Teaching. Educational Systems Department, System Development Corporation, Santa Monica, California 90406.

Typically, the use of MENTOR, or of other languages in Table 2, is encouraged to provide in the computer program a set of conditional statements which, for any stage of discussion, makes the computer's reply dependent not only on the student's current inquiry or assertion, but also on the history of the conversation. Because history is stored almost automatically, and complex conditional expressions can be written with considerable ease, it is convenient for describing a dialogue which is conditional on the present context and the history of discussion with each student. PLANIT and MINORCA have other distinguishing characteristics for specific areas of instruction and techniques of learner assistance.

I do not mean to imply that other languages in Table 1 cannot be used for programming a dialogue or conversation. Indeed they have, but with considerable author inconvenience and programming expense. On the other hand, each language in Table 2 has one or more characteristics which have encouraged this style of instructional programming.

The example given in Figure 2 is part of a mystery problem coded in a notation similar to the MENTOR language. Such languages have been used to specify exercises for training skills (such as information-gathering and decision-making) needed in medical diagnosis or electronic trouble shooting. Considerable convenience is gained by providing a convention for "stacking" of replies so that the line marked "1)" is used the first time the student reaches that point in the exercise, "2)" the second time, etc. Also the directives to the computer for sequencing parts of the conversation (for example, "IF ALL REPORTS, DO LAB") appear very much like logical expressions.

4. Description of a standard procedure by which material is presented

Some of the languages which have been used for writing standard procedures (to be applied to various files of content) are listed in Table 3. All of the languages or notations in the first two categories had to be programmed for the computer in a regular computer language which could be interpreted by the machine. Any of these general-purpose languages could have been used directly for instruction, but it has been found desirable to produce simpler languages tailored to specific instruction tasks. Some general-purpose languages are especially convenient for writing procedures for interactive use on a computer, or in particular, for producing formats for conversational instruction.

For some time, programmers using CATO at the University of Illinois have prepared various teaching logics or basic strategies into which curriculum authors can place their material. A PLATO tutorial logic provides the most simple and convenient "language" I have seen, so much so that it is better called a "data format" or set of conventions for preparing a file of curriculum materials. An example of such a file, given in Figure 3, refers to the same drill exercise programmed in Figure 1.

This simple notation or data format is a straightforward approach to serving the needs of an author; it provides a format into which he places elements of the curriculum: questions, answers and hints to be delivered to the student in sequence. Because of the convenience of the video terminal on the PLATO III system, each question and corresponding hint was placed in the appropriate location on a large sheet of transparencies to be inserted in a

(continued page 27)

```
GENERAL  "Proceed with investigation."
     ACCEPT
          IF /suspects/
               1) "Wife, brother and partner."
               2) "No new suspects."
        IF /lab, rifle, glass, pipe/
               IF ALL REP, TO LAB
               "I advise you to check reports first."
        IF /interrogate/
               IF ALL LAB, TO INTERR
               "I advise you request lab tests first."
          ...
          ...
          ...
          "I don't understand."

LAB  "This is the lab."
          IF /glass/
               IF WIFE
                    "Glass contained arsenic."
               1) "Prints belong to the wife."
               2) "Nothing new."
          ...
          ...
          "What is it you want?"
          ACCEPT
          TO LAB + 1
```

Figure 2: Sample of a notation suited for exercises in decision-making and providing for conversation within a limited context.

TABLE 3: Description of a procedure by which material is presented:

CATO: Compiler for Automatic Teaching Operation. Computer-based Education Research Laboratory, University of Illinois, Urbana, Illinois 61803.

TSA: Teacher-Student ALGOL. Institute for Mathematical Studies in the Social Sciences, Ventura Hall, Stanford University, Stanford, California 94305.

ISL-1 and ISL-2: Instructional Systems Language. RCA instructional Systems, 530 University Avenue, Palo Alto, California 94301.

SKOOLBOL. Learning Research and Development Center, University of Pittsburgh, Pittsburgh, Pennsylvania 15213.

XXXX: Unnamed extension of FORTRAN. Human Learning Institute, University of Minnesota, Minneapolis, Minnesota 55455.

FORTRAN, ALGOL, MAD, SNOBOL, TRAC and others useful for programming instructional procedures, although not prepared specifically for that purpose.

---

scanner. The computer program successively presents the question frames, provides a hint when the student asks for it, provides the right answer when needed, and records performance data for later inspection by the author of the exercise. The separation of content and procedure limit the flexibility of the author (once he has committed himself to a particular procedure) but increase his output.

More recently CATO was used for the preparation of a higher level language called TUTOR, which is somewhat like COURSEWRITER. Since CATO is an extension of FORTRAN on a CDC machine which allows assembly language statements to be interspersed, an expert and experienced user of the system has available programming capability from assembly language (close to the basic machine capabilities) through procedure-oriented to problem-oriented languages. Without the backup provided by PLATO system programmers, however, the author using TUTOR still may be obstructed by the implicit assumptions and limiting conventions of any one "author language".

Instruction Sequence	Questions	Hints	Answers
1	$2^3 = ?$	$2^3 = 2 \times 2 \times 2$	8
2	$3^2 = ?$	$3^2 = 3 \times 3$	9

Figure 3: From a simple program (data set) which is presented by a general procedure program

Another language for writing procedures is RCA's Instructional System Language: ISL-1 was adapted from Stanford's Teacher-Student ALGOL (TSA), and ISL-2 is a modification of BASIC. The major use of ISL-1 has been to represent the procedures for mathematics and language drills in the Stanford project. In fact, the RCA instructional system operating in the New York City Schools was particularly arranged for economical math drills for large numbers of students (150 to 200 simultaneous users). The system in use in the Waterford Schools in Pontiac, Michigan, allows a greater range of instructional programming using ISL-2, but is not likely to support as many users if the programming options are exercised.

Figure 4 contains a procedure for assembling a drill exercise from a file of items such as that suggested by Figure 3. For this example I have used a hypothetical language which includes some aspects of CATO and some of ISL. This drill procedure could be made more general and powerful, i.e. it could be arranged to produce a greater variety of drill problems for the time invested by the curriculum designer. For example, other procedure programs have been designed to specify that numbers for the problems be selected at random, although within specified ranges, and so that other problem characteristics such as regrouping (i.e. "carry" or "borrow") are maintained.

My idealised notation may make such procedure writing appear accessible to curriculum experts having little acquaintance with computers. However, in reality, such programming requires persons expert in the particular skills of programming and computer use. There is no easy road to construction of complex, computer-based learning strategies.

5. Specification of an environment for programming and problem solving

Interactive programming languages are already available to users of general-purpose, time-shared computing systems who do not have access to dedicated, computer-assisted instruction systems. I have found that the languages listed in Table 4 provide many of the features desired by authors preparing materials for instructional use of computers. Furthermore, the interactive mode for program construction emphasised in some of these languages provides greater convenience for construction, debugging, and alteration of programs than is characteristic of CAI systems. Immediate diagnostics and error recovery procedures allow mistakes to be corrected when discovered; direct-mode execution of statements is useful for displaying parameter values and for restarting at any point within the program during testing. PLANIT and FOIL have some of these features for interactive program preparation and testing.

The computation facilities found in problem-solving languages are useful in many learning exercises, and convenient computation is conspicuously absent from most languages designed especially for computer-assisted instruction. To remedy this shortcoming, some of the languages in Table 1 (notably PLANIT, PICLS and CHIMP) have built an elaborate "calculation mode" or linked the "author" processor to a "computational" language such as BASIC.

Authors of computer-based learning exercises at the University of Michigan have used languages which are convenient for students in the design of exercises, a factor which is especially important when the author is using mathematical models and simulation. Typically, students begin in a tutorial mode, then shift to exploration of some underlying model the professor has designed; some have progressed to the point of using the computer as a research tool and model builder, regardless of their previous experience with computers.

```
         For I = 1 TO N UNLESS ERRORS > 5
             BEGIN
             HINT = 0
NEW          ERASE
QUES         DISPLAY QUES(I)
             ACCEPT UNTIL TIME > 20
                 BEGIN
                 IF RESP = ANS(I)
                     TYPE "Correct"
                     INCREMENT I
                 ERRORS = ERRORS + 1
                 IF HINT = 0
                     DISPLAY HINT(I)
                     HINT = 1
                     TO QUES
                 DISPLAY "The answer is" ANS(I)
                 INCREMENT I
                 END
             IF TIME UP = 0
                 DISPLAY "Time is up; try again."
                 TIME UP = 1
                 TO NEW
             DISPLAY "Time is up."
             DISPLAY-PROCTOR "Too much time." STUDENT, LESSON
             TO HELP
             END
             IF I = N, TO NEXT
             DISPLAY-PROCTOR "Too many errors." STUDENT, LESSON, I,
                                                             ERRORS
HELP DISPLAY "You seem to need help. Ask "ON-DUTY" for
                                                 assistance."
             PAUSE
```

Figure 4: From a program stating a drill procedure for the data set in Figure 3.

TABLE 4: Specification of an environment for programming and problem solving:

ACME: Advanced Computer for Medical Research. Real-Time Computation Facility, Stanford University Medical School, Stanford, California 94305.

ALCOM: Applied Logic Computing. Applied Logic Corporation, One Palmer Square, Princeton, New Jersey, 08540.

APL: A Programming Language. Education Research, T.J. Watson Research Laboratory, Yorktown Heights, New York 10598. Commercially available through IBM and a number of time-sharing services.

BASIC: Beginner's All-purpose Symbolic Instruction Code. Computer Center, Dartmouth College, Hanover, New Hampshire 03755. Commercially available from GE, Tymshare, UNIVAC, IBM, and others.

BRUIN: Brown University Interactive language. Brown University, Computing Center, Princeton, New Jersey 08540.

CAL: Conversational Algebraic Language. Computer Center, University of California, Berkeley, California 94720. Commercially available from Com-Share, XDS, and others.

CITRAN and REL: CIT Translator; and Readily Extensible Language, Computing Center, California Institute of Technology, Pasadena, California 91109.

FOCAL: Formulating On-Line Calculations in Algebraic Language. Digital Equipment Corporation, Maynard, Massachusetts.

IITRAN and CALCTRAN: IIT Translator; and Calculating Translator. Computation Center, Illinois Institute of Technology, Chicago, Illinois 60616.

ISIS: Irvine Symbolic Interpretive System. Computer Facility, University of California, Irvine, California 92664.

JOSS: JOHNNIAC Open-Shop System. Computer Sciences Department, RAND Corporation, 1700 Main Street, Santa Monica, California 90406.

LCC: Language for Conversational Computing. Computation Center, University of Pittsburgh, Pittsburgh, Pennsylvania 15213.

LOGO. Department of Educational Technology, Bolt Beranek and Newman, 50 Moulton Street, Cambridge, Massachusetts 02138.

PIL: Pittsburgh Interpretive Language. Computing Center, University of Pittsburgh, Pittsburgh, Pennsylvania 15213.

POP-2. Department of Machine Intelligence, University of Edinburgh, Scotland.

QUIKTRAN; "Quick" FORTRAN. Information Marketing Publications, International Business Machines, Monterey and Cottle Roads, San Jose, California 95113.

RUSH: Remote User Shared Hardware. Allan-Babcock Computing, Los Angeles, California 90067.

TELCOMP, STRCOMP and ISRCOMP. Bolt Beranek and Newman, 50 Moulton Street, Cambridge, Massachusetts 02138.

TINT. System Development Corporation, Santa Monica, California 90406.

An excerpt from a sample program, given in Figure 5, uses the Pittsburgh Interpretive Language (PIL) to set up a situation in which beginning students practice managing a simulated fish population and manipulating the underlying ecological model. The first part of the program initialises the parameters of the model; the second allows a proctor or teaching assistant or other student to reset the parameters to values of his own choosing; the third part generates a "history" on which the student makes his first decision. The model itself is invoked within a conversational sequence which represents a simplified management situation. The program is concluded with a provision for a short exchange with the student about optimum strategy and level of management.

6. Relative use of the four kinds of programming

More languages and dialects fall in category one than in any other, and probably more author hours have been invested in the computerisation of programmed instruction text than in other modes of instructional programming. The "data formats" and "drill procedures" have been used extensively at installations that have that kind of language facility.

Increased use of procedure-statements with separate curriculum files will be beneficial for the field, and large curriculum development projects using computers will require this approach for economy. I must say again that languages of the procedure-writing type are intended for computer programmers and for educational technologists specialising in computer applications; these persons should produce the user-oriented languages or data formats which then provide maximum convenience for the curriculum expert.

Some data formats may take on special and interesting characteristics: the curriculum design team can represent the desired knowledge and skills in some kind of structure which both they and the computer can interpret; and specialised computer programs will be set up to attempt, through various means built-in by specialists in learning and information systems, to ensure that each student achieves those objectives.

More generality would be achieved by preparing computer programs which assemble instructional materials from elements of the subject matter and relationships among these elements. Availability of such procedures would permit the author to describe an entire class of problems by one set of statements. From one general description, any desired number of test or instruction items could be generated for presentation to each student as needed. A procedure which assembles or generates materials is likely to have more possibilities for individual adaptation than one which selects successively or branches through a large pool of specific items.

```
*Control is described in part 1 (with part-step numbers at left
margin)

1.0 FOR peak=11: FOR max=30: FOR run=18: FOR history=7: FOR
model = 6  STOP.

*proctor may reset parameters as desired before typing "go"

1.1 TYPE "Read instructions: Regulation of a salmon fishery."
1.2 DO  part 2.

*initialisation

2.1 FOR entire=0: FOR total=0: FOR cycle=0: FOR year=1969: SET
error=the time.
2.2 FOR med=.5*(max-peak): FOR low=.5*med:
2.3 IF history=0, TO step 1.3.
2.4 TYPE "History:".
2.5 TYPE "year  catch  escapement".
2.6 FOR escape=peak/2: FOR y=year-history TO year: DO part 3.

*generate history

3.1 FOR catch= med+escape*random number of error BY -.2*catch WHILE
run-catch= 0: next catch.
3.3 SET escape=run-catch.
3.4 TYPE in form "# # # # ___.___.___.___", the BCD value of
year, catch, escape.
3.5 DO part model.[compute run for next year]

2.7 SET history=0.

*management control

1.3 TYPE "By which strategy will you manage fishery: escapement or
catch?".
1.4 DEMAND in form "############", strategy.
1.5 TYPE in form "####", the BCD value of year.
1.6 IF the first character of strategy="e," DO part 4; ELSE DO part 5
*
4.1 DEMAND escape
4.2 IF escape > 0, TO step 4.4; ELSE TYPE "You must let some fish
escape.".
4.3 TO step 1.1.
4.4 FOR catch=run-escape: IF catch >0, TO step 4.6; ELSE SET
catch=0.
4.5 SET escape=run.
4.6 TYPE in form "catch= ___.___", catch.
```

Figure 5: From a program providing an environment for simulated management and exploration of models.

Interactive programming languages are much more widely used in instructional exercises than is generally recognised by persons working with the "author" languages in my first two lists. Use of interactive languages will increase because they are accessible and convenient for instructional exercises and will be used for computer literacy and technical skill courses anyway. One time-sharing service in north eastern United States offers schools a terminal device, unlimited use of the system 24 hours a day, and a few thousand characters of storage for about $350 per month.

Some of the enthusiasm for conversational computing may be attributed to non-essential features: quick response and understandable diagnostic messages can be provided also in remote-entry batch systems. Now that commercial services are being offered to (and purchased by) public schools, it becomes increasingly important to isolate the essential contributions of interactive programming languages, and to determine effective cost conditions. However, I am convinced that five years from now, exploratory use of those languages for instruction will have contributed more to education than have similar trials of frame-oriented programming in the last five years.

III. RECOMMENDATIONS

1. Maintain Flexibility

All research and development projects and most operational ones should remain flexible for a time yet, avoiding restrictions placed on users by any one programming logic because of the language and system. Requirements as perceived by users continue to change, and the language characteristics can follow along if not fixed rigidly at the start.

Users should not be restricted to a single, preset logic such as the basic Coursewriter or Planit, or the math drill strategy of the RCA IS/70. Different disciplines and teaching objectives determine different approaches to computer-aided instruction, and variety in author preferences can also be justified. If the language capabilities do not match the needs, important human resources are wasted trying to make a system and language do things beyond its intended scope.

New logics (or data formats or task-oriented languages) should be able to be prepared readily in response to the needs and suggestions of potential users. The computer skill required to adapt or extend a general-purpose language to suit a particular instructional programming task is likely to be considerable, but the time required will not be great if the basic language is suitable. New logics, even new languages, are prepared with relative ease using the PLATO compiler at the University of Illinois Computer-based Education Research Laboratory. Similar work is possible on some time-sharing systems, although flexibility in software and quick response of system programmers to user requests is not common outside the universities.

Probably the best way to meet the requirements placed on a system and language today, especially for a research-oriented project is to work within a general-purpose system providing more than one suitable programming language. At least one language or application program should provide a format for preparing computerised programmed instruction: FOIL, LYRIC, and a number of others can be installed quite readily by compiling a specialised processor programmed in a common language such as FORTRAN. Others should provide for string processing (e.g. SNOBOL), procedure writing (e.g. ALGOL) and interactive problem solving (e.g. JOSS). Once a lesson designer determines what approach is likely to be successful with his students and teaching goals, that approach should be made more economical, and perhaps more convenient, through implementation of a specific language or notation.

For the project already committed to a restricted system, e.g. the IBM 1500 does not include much range in programming capability, the best approach may be to prepare instructions using some more appropriate notation and translate automatically on another computer into the code needed, in this case Coursewriter II. Although this should ease the task of the lesson designers and manual coders, the programs still will be no more economical than allowed by the Coursewriter II processor and system. The manager of facilities for a school or research project should have some options from which to select the best programming language for each purpose and perhaps a better machine.

Ideally the creative curriculum designer would work completely free of system and language constraints, describing materials and conducting trial student use in some general notation and non-computer format. Later the exercise would be implemented in the language judged by an assisting programmer to be the most practical. A general notation evolved through use of this approach at the University of Michigan. Each author adapted it to his purposes; and programmers implemented as best they could the author's intentions as expressed in this communication language, sometimes using differe computing systems as well as different languages.

In reality the creative curriculum designer may find even the best among available computer terminal devices and data structures too restrictive. He should be encouraged to carry out limited use of procedure-oriented learning exercises with non-computer formats and human teacher aids (other students and paraprofessionals). The ultimate of flexibility in a computer-related instruction project is to be quite free to leave the computer to achieve project goals, e.g. instruct students more effectively or discover important factors of individual learning, whether or not assisted by the comput

2. Adapt to Specific Uses and Users

Clearly the requirements are different for instruction, testing, counselling, curriculum development, research on instruction and learning, or research on languages and systems. No one language and system support package can be expected to serve all uses and users. However, by careful selection and extension the available language capabilities can be matched to specific requirements.

Differences among the needs and interest of users shape the system support features. The author needs control of computer processing capabilities and information storage, and he may wish to pass on this control to the student or other user of his instructional procedures and information files. The system should at all times be interpretable to all users and especially the student. When the automatic processor gets lost and doesn't respond intelligibly, a learner who actually was on the right track may inappropriately blame his own work, and his performance and attitude will suffer.

Actually any single user may take on different roles, and should not be unduly restricted by the language and system. The author should be able to move easily from curriculum development to research and back, using the computer for data processing, modeling and information management in pedagogical as well as scholarly work. An interested and able teacher will be an author of new materials as well as a manager of instruction.

In particular the student will take on different roles. He may begin in a tutorial exercise, move into the role of a practitioner in a simulated exercise, then examine the model as a researcher, and finally, as a teacher, set up new conditions for the simulation to be used by another student.

3. Exploit Interactive Mode of Computer Use

On-line conversational use of computers is almost certain to be more costly than less glamorous means of access to information processing aids. However, the occasional user of computers is more likely to benefit from the interactive mode of operation than experienced or frequent users. Programming languages and learning exercises should be selected to take advantage of opportunities for the infrequent learner-user to carry on a dialogue with the system.

When the student doesn't know where he is in the system or exercise and cannot determine what to do next he should always be able to get a useful reply by questioning the system, and he should be able to determine a suitable place to resume the exercise. When the system begins to display material in greater detail than the student can use, he should be able to interrupt and specify another mode of display or a more suitable level of detail.

The learner should feel encouraged to test tentative ideas and try out possibilities, knowing the system not only will permit such explorations, but will help them to be successful. Suitable computer programs will keep track of loose ends while the user is sketching in ideas, accept details later, and provide immediate and interpretable reply when the user's instructions are ambiguous or incomplete.

4. Relate Language Maintenance and System Operation to Project Goals

For a system serving authors, teachers and students in day-to-day operations, clear documentation of the languages and user support features is essential for prompt and effective maintenance. Errors in the processor programs will occur; minor additions will be required; and occasionally a major modification is justified. Reliable service and an environment for gradual improvement will follow from sound documentation.

A log of system use and maintenance is a useful management tool. The record of use, problems encountered, and goals achieved indicate the extent to which the operation is serving the primary users; the log also provides a basis for projecting cost and elapsed time required for future improvements. Such information is especially useful in an experimental system which continues to change.

Maintenance of accurate reference manuals and effective training materials has high priority for project staff if the project goals emphasise use of the system. Those expert in system design and use will spend considerably less time trying to communicate to naive users if the project purpose is exploration of language capabilities. Documentation of computer-based learning exercise should receive priority attention by the designer to the extent the project plans to use the exercises in the future or to promote use by other individuals and institutions.

Criteria for effective system operation can be made explicit, for example, by assigning a monetary value to time lost due to faults in a program, modifications of a language processor, or change over to a new system. Such designations will make explicit the relative priorities of student use, curriculum development, educational research or system experimentation.

IV. STATEMENT OF POSITION OR POINTS OF VIEW

1. ASSESSMENT OF A LANGUAGE AND SYSTEM

A. Representations for Language and System Characteristics

The forced juxtaposition of two or more languages, whether by a list of aspects, characteristic samples or measures of author performance and satisfaction, cannot help but encourage each designer to improve the capabilities of his language at least for those purposes represented in the comparison.

1. Functional aspects compared against a standard list

The comparison of languages by the 60 "common aspects" in the Educom comparative study emphasised similarities by presenting together the way in which 40 different languages would be used to accomplish the same function. Such discussion prompted some languages designers to fill in a few blanks in the columns describing their languages, that is, they added to their own language some of the capabilities previously described only for other languages in the comparison table.

A summary table arranged by common aspects cannot be complete and free of error: the languages are changing rapidly; the designers are slow to provide current documentation; first-hand programming experience in each language is not possible. Different approaches to summarising favour one language or another; and more important, different approaches to instructional use of computers require essentially different language characteristics.

Languages explicitly intended to serve different instructional programming tasks should be described for purpose of comparison by different sets of attributes in different tables.

In other words, programming tools should be grouped with others of similar purpose when making relative comparison, rather than thrown together with all the tools of very mixed purposes.

When making decisions about languages and systems, the relative weighting of various criteria must be determined by each project or user upon considering: a) the age and background of the student or other users; b) the relative importance of research, development, implementation and operations; c) the relative interests of project staff in general system characteristics, programming languages, or instructional materials; and d) the availability of funds and of a general-purpose system.

Some standard format or common notation is needed for writing an individualised description of each language so that its characteristics can be interpreted readily by interested persons who did not participate in design of the language. In order to communicate with potential users of computer-based systems, a notation for description should be readily interpreted by those who have little experience with languages and systems. (For example, BNF is not suitable!)

2. Samples of code, presumably typical of user and task requirements

The best test of a programming language is through use, and the closest approximation for a reader not yet familiar with the language is a sample of use on standard test situations.

Each language has unique features, and any small number of test programs will favour one or another. It is difficult to represent the capabilities of any language in a few pages of sample programs.

Many of the languages continue to be changed, and the samples obtained one month may not be characteristic of what is being done with the language six months later.

3. Empirical measures of usefulness: programming time, errors, attitude, execution time, etc.

Efficiency of a language sometimes is measured by the number of machine language instructions, source language instructions, characters in the file, etc. However, this measure depends on the instructional strategy employed, the propensity of the author for writing the same learning task description in a smaller number of instructions, and his concern for providing responses for a number of rather unlikely eventualities.

A major problem for evaluation of any component of an instructional system is the definition of a measure of accomplishment which avoids reliance on how long the student spends with the new learning materials. The efforts of a programmer-author should show in concepts acquired by the student (or skills perfected), not just student time at a terminal, as if the system were baby-sitting.

An effective system will encourage curriculum designers to exploit the computer medium to improve and expand the content and skills taught, and to use the occasion of revision to drop some material which is obviously useless.

Criteria for judgments about languages should be more explicit. Words such as reliability and flexibility are used as if everyone agreed on what they mean; in fact very different measures of the implied concept have been employed. It is not necessary that all writers agree upon any single definition for a term or a unique measure for a criterion, but each writer should make his use of terms and measures more explicit.

Journals should adopt a firm editorial policy which requires clarification of the referent or measure of "power", "elegance", and other such terms when used in published reports.

B. Suitability for the Primary Users, Considering Background and Goals

User performance (e.g. error rate) is an important consideration in selection (or design) of a language. Increased training may not be the solution; programming errors which appear frequently in instructional programs can be reduced by changes in the translator.

One should not always blame the user for programming errors but look at factors in "reliability" of the semantics and syntax of the language. The same applies to reliability of the instructional program. Some of the errors which may occur during instruction and interfere with learning by an individual student should be blamed on the author, or the language designer, etc. not on the student.

1. Student

Students need to be able to get information about the system operation and procedures at any time: Is it operating? Why was his message not accepted? How long might he have to wait before starting a certain exercise? Procedures should be simple, including conventions for erasure within a message or cancellation of entire blocks, indication of availability of a device for input, etc.

The student of a particular discipline should not have to acquire computer skills and conventions which are unnecessary for his study, e.g. complicated keyboard skills or new notational conventions unrelated to the subject of study, and necessary only to reply to a computer tutor.

The information processing capability of the computer should be as available to the student as it is to the lesson designer or the researcher.

Computers and programming are now becoming part of everyday life, and the designer of a computer-based lesson should not hesitate to require of the learner certain computer skills otherwise unrelated to learning in that subject area.

2. Instructor (supervisor or manager of learning)

The system (and language) should accommodate the instructor, to the extent that he is expected to adapt the learning materials for each group of students and his particular style of teaching.

The right data on student performance and attitude should be available to the instructor at the right time and in the right context; relevant, timely and interpretable information is essential for effective management of learning.

If the system is designed to run without intervention of classroom teachers or other supervisors, then computer memory space and processing time should not be wasted on features included only for these personnel who do not use the system in operation.

3. Counsellor or administrator, if different from the instructor

Management working from a perspective different from that of the teacher may require data in a somewhat different format and context: the counsellor needs detail on individuals and in the context of other work or plans of that individual; the administrator needs detail on use of resources (personal and technical) in the context of the total instructional system.

An on-line data management system for school records would more than pay for itself in saving administrative time and reducing errors in quick judgments.

Most of the decisions which are made in educational systems do not justify on-demand access to current data; decision points can be anticipated, and many of them are periodic.

4. Author or lesson designer

Staff on a curriculum development project require convenience and predictable operation for writing and testing exercises; these requirements may conflict with the economy and convenience required for day-to-day student use in the schools. Terminal devices provided authors are more expensive, the speed of compilation of new programs is more rapid, priority is given to revision of materials, etc.

Data obtainable from student use of learning exercises may be selected and arranged differently for the purpose of revision of the exercise (by the author) than for assessment of student performance (by the teacher or administrator).

Control should be left with the author or lesson designer. As examples: the user may prefer off-line to on-line entry with immediate diagnostics; he may wish to establish some of his own notational conventions rather than always to adopt those of the system programmer who designed the language; he may wish to change the standard replies (such as from "wrong, try again" to "not recognised; try again"), or to adjust the tolerance for accepting mis-spellings or typographical errors in otherwise correct answers.

Although there are many tricks that can be played with the counter registers and character registers of "author" languages, the lesson designer must apply peculiar commands for manipulating these rudimentary elements of information processing by computer. Playing these games will distract otherwise effective authors from their primary purpose: helping learners in some efficient fashion.

A procedure for presenting curriculum materials should be prepared by expert programmers according to a design developed by a team of subject experts and educational technologists; then the writers enter material into a system which in part can protect them against their own errors.

5. Researcher on instruction and learning

An educational researcher is willing to pay much more per terminal hour than an educational administrator, if the system provides the required facility for stimulus presentation and data recording.

Research uses usually require more detailed data than teaching and curriculum development, and some data are unrelated to teaching purposes: latency, physiological measures, etc.

In some research uses the computer makes no contribution to learning by the subject during the experiment; the researcher need not be concerned about computer-based instruction contributing in some way to the learning of the student beyond what would have been achieved without the computer.

6. Researcher on the systems and information sciences

A project on language characteristics and system features must invest in flexibility, even at the expense of author or student convenience. The curriculum writers who choose to work with such a project must be willing to give up convenience for the sake of experimentation, e.g. adjust to language changes, accept errors and unreliability, and modify or discard outdated programs.

An experimental system is entirely different from an operational one: relative costs, responsive to different users, etc.

Some day the experimental and operational purposes can be brought together in the same system.

C. Suitability for the Mode of Computer Use

1. Routinised drill and testing

A system used for drill and testing should have a library of standard routines which can be adapted for whatever pool of drill or test items the user might like to introduce into the system.

A user should be able to add readily to the library of routines or procedures for drill and testing.

The description of data (test items) for standardised routines should be straightforward and convenient for the author.

2. Computerised programmed instruction

A system used for presentation of programmed instruction materials should not require of the author much more than a specification of the text materials as they might be presented in booklet form rather than on the computer.

When variety is required it should be introduced at random or according to parameters under the author's control, e.g. selection from a set of confirmatory replies or options to introduce review material.

Frame-by-frame writing of programs with an author language is on the way out. A few years from now less than one tenth of any computer-based course will be programmed by an author or his technical assistant directly in languages such as COURSEWRITER and PLANIT.

3. Diagnosis and remediation

A system intended to provide individualised attention to learner difficulties must have some generalised procedures to apply each time an answer is incorrect; full programming of each frame of a diagnostic test for all possible student difficulties is not feasible for the major part of self-testing and remediation exercises.

Diagnosis and remediation are important uses for computers in instruction, for the student is likely to benefit greatly from the opportunity for interaction with a prepared procedure. This mode of use places correspondingly greater demands on the programming language employed.

Standardised learning materials can be presented effectively by other media than computer-based systems. The more expensive information processing devices and programming languages should be applied to those situations such as remediation where individualisation is not only desirable but necessary.

4. Question answering

Being a device for storing, processing and retrieving information, the computer should be programmed to assist the individual learner in his own scholarly endeavours by providing answers to questions about information sources, fact, etc.

One general approach must be applied to many topics and learning exercises if questions answering systems, being expensive to prepare, are to be practical.

The expensive systems, designed to respond to inputs of great variety and be applied to a wide range of topics, become practical when the field of inquiry and the format for questioning are suitably restricted.

5. File and text manipulation

Handling files and strings of text is not a process incidental to computation, but a substantial part of information processing sciences. In the educational setting this mode of use should be accorded full attention in the library of programming languages made available.

The lesson designer (or the student as a direct user of file information) should not have to manipulate textual information with primitives applied only to characters and lines. Languages should allow suitable representation for units such as words, sentences, paragraphs, and chapters as well, and for search and transformation operations.

6. Numerical problem solving

A conversational problem-solving language such as APL, BASIC, or CALCTRAN would be much more successful on a regional computing service than an author language like Coursewriter. The development of high-quality tutorial instruction requires a major commitment of funds and personnel, including instructional designers, media specialists, and programmers not usually available at remote locations. Typically, one person at a site, almost never on a full-time basis, must give demonstrations, teach programming and consult on applications. An interactive computing language can be more easily taught and more effectively used with limited resources.

The problem-solving mode will be over-valued and misapplied, as was COURSEWRITER five years ago. However, more instructional materials of significance are likely to survive in this mode in the next five years, than have been seen in the computerisation of programmed instruction in the last five years.

Undirected use of a simple programming language is not always a cost-effective way to develop skills in problem solving or conceptualisation of procedures. The designer of a learning exercise or environment must consider adding language features specific to the tasks the learner is to carry out, and try to describe ways of assessing the learners progress along any path to a solution, perhaps specifying interruptions to provide information about difficulties encountered.

The special contributions of interactive mode of use to student programming and problem solving are not obvious. Much of what is said to be unique to interactive processors can also be accomplished with well-conceived compilers in a system providing very quick batch response. /See VI below on interactive mode contributions./

7. Graphic display

Although graphic capability is much sought after by many computer users presently restricted to alphanumeric displays, those who do have the technical capability to show the student line drawings and accept simple sketches in return find the associated programming task horrendous. Programming problems in this domain have not been solved for instructional users.

8. Other modes not defined (i.e. growth potential to meet unanticipated user needs)

Other modes of use may not be included in the listing above, and many new uses are yet to be contrived. Each places special demands on the programming language and system which should be met if teaching and learning are to proceed in an efficient and effective way. Computers and information processing should be at the disposal of the learner and others in the educational system, and programming languages should be adapted to their purposes.

No language can be expected to have all those features which may be desired by various users during its lifetime. Most languages provide for definition of subroutines (separate routines designed for repeated use) with a transfer statement which saves the present location (or any designated location) so that control can later return to one of the saved locations. Macros provide another way to avoid repetition in coding by packaging a number of statements to be called on by one statement, in some cases with parameters. The facility for adding new operations or statements is less common but potentially very significant.

Some languages allow the programmer to write special functions, perhaps in another language, with a list of arguments automatically transferred from one to another. Ideally, programs written in any other language could be linked to instructional program so that data could be passed from one to the other when the student moves from one to another (e.g. from tutorial to a special simulation or model building package).

D. Suitability for the Style of Program Preparation

If the programming language capabilities and conventions are not matched to the instructional programming task, exploration of new curriculum objectives and learning techniques will be suppressed and large scale development of materials will be discouraged.

1. Description of successive frames or items

More instructional programming has been done by preparation of frames than any other approach, and most special-purpose "author" languages provide well for this style. However, additional provisions for establishing normal modes of operation or calling on standardized procedures would reduce unnecessary repetition in the instructional programmer's task.

The most straightforward approach to serving the needs of an author may be to provide a format into which he places elements of the curriculum. The computer program successively presents the question frames, provides a hint when the student asks for it, provides the right answer when needed, and records performance data for later inspection by the author of the exercise.

The frame-oriented description of testing or instruction is a kind of computerized programmed instruction. The similarity of the code and conversation to a programmed text is apparent. In fact, translators have been written to accept linear (or simple branching) programmed text and derive CAI interaction with a student.

2. Provision for conversation within a limited context

Authors should be able to compose complex, conditional procedures more easily than at present; MENTOR and PLANIT include good examples of convenient conditional expressions by which a procedure can be made dependent on student performance. Most authors of computer-based instruction have made little use of computer logic and memory, perhaps because they are unable to conceptualise complicated sequencing rules, or because they are quickly discouraged from doing so by the clumsy syntax of programming languages prepared for them.

Programming for conversation in relatively unconstrained English may not be a reasonable approach until some breakthrough in research on processing natural language provides an efficient and reliable means for "understanding" or at least classifying what the student says.

3. Description of a standard procedure by which material is presented

Content should be prepared in a form independent of particular computer conventions and convenient from the viewpoint of a context specialist. The control procedure which administers a learning task should be free of specific content material. The answer processing and other conversation-handling aspects of control should be separate from the scoring and sequencing algorithms.

Computer programs which assemble instruction materials from elements of the subject matter and relationships among those elements should permit the author to describe an entire class of problems by one set of statements. From one general description, an indefinite number of test or instruction items should be generated for presentation to each student as needed. A procedure which assembles or generates materials is likely to have more possibilities of adapting to the individual than one which selects successively or branches through a large pool of specific items.

Increased use of procedure-statements and (separate) curriculum files will be beneficial for the field, and increasing use of computers in large curriculum projects will require this approach for economy.

Procedure-oriented languages are for computer programmers and for educational technologists specialising in computer applications; these persons should produce the user-oriented languages or data formats which maximise convenience of the curriculum expert.

One could have too large a library of strategies and too much individuality among students and topics for standardised techniques to be useful.

Until instructional objectives for a topic are rather well defined (for example by standard procedures for testing use of facts, concepts and simple skills), development of prescriptive curriculum for individualised instruction in that topic is not likely to be successful.

The practical application of standard procedure programs and generative techniques applied to curriculum files on any specific subject area or training situation raises many questions: How are information structures to be described by the subject expert and stored in the computer for use in such procedure statements? How are materials to be assembled according to general rules? How is input from the student to be processed in some general way which determines a suitable reply? Can patterns or sequences be identified which prescribe certain adjustment for the student on succeeding learning experiences?

4. Specification of an environment for programming and problem solving

If an on-line problem solving language is suitable for simulation and model building, then that language certainly is of interest to designers of computer-based learning environment. First, the subject expert may build models on which to base games or simulated practice for students to try. Second, he may guide some students through revision of the models and construction of new ones. In general, he wants to show students how to use the computer for information processing in his discipline; as lesson designer he might produce a "mentor" which advises each student on how to get maximum value from the computer as a problem solving and scholarly aid.

The most significant contribution of simple, interactive programming languages may be through increased student use of computers for problem solving and scholarly endeavour on individual initiative.

5. Implementations Available: Machines, Memory Size, Costs, Reliability, etc.

Variations among machines, even different models of the same machine, will affect the language features, efficiency of operation, number of users, and even the kinds of use.

Knowing that a language processor is available for a particular machine, say an IBM S/360 Model 50, is not enough. The specific configuration (computing resources) assumed by the language designers must be detailed: amount of core memory, special features such as memory protection, number of disk and tape drives, terminal controllers, etc.

Different implementations of a language processor, even with the same functional specifications and for the same configuration of the same machine, will vary in processing capacity and cost of use.

Assessment of the reliability of a particular implementation of an instructional language and system should consider the rate at which new errors had been appearing as well as the number of presently known errors. Although a programming system might be delivered with all known "bugs" fixed, the continuing appearance of three new ones each week thereafter would hardly be tolerable.

F. Documentation, Teaching Aids and System Maintenance Available

Complete and interpretable manuals are essential for various users. Such reference materials can be incorporated in the processor (computer programs) to be printed out on request or as they appear to be needed, but in the past such an approach has been expensive and incomplete. Computer-based manuals continue to be attractive, especially for the experimental language which is continually being changed, making difficult the maintenance of current information in printed formats.

An introductory manual or primer for a language and system reduces the need for costly live instruction to initiate new users. Primers have been written to be used while working at the terminal of an interactive system, inviting the reader to test each new convention or concept as it is described in the text. Such self-instruction has also been presented by films or video tapes at somewhat greater expense and lessened convenience.

Adequate documentation of system programs often is lacking, making maintenance or improvements very costly or impossible (without reprogramming large sections of the processor). An institutional user should be satisfied it has description for its systems programmers, or a tight contract for maintenance from the software supplier.

2. GENERAL IMPLICATIONS OF A LANGUAGE AND ITS IMPLEMENTATION FOR STRATEGY OF INSTRUCTION

A. Data Available for Automatic Decisions

If the system is not capable of measuring the time each student takes to respond, and making this available for decisions at the moment as well as later, then the lesson designer is denied this data for his instruction strategy.

Performance and preference records accumulated one day should be available the next day or the next month from some strategies of instruction. Records of individual learning characteristics may be more significant in selecting or arranging a later learning experience, than in phrasing the next question or diagnostic within the same exercise.

Some lesson designers have wished to pass information from one student to another, or provide summary information for all, whether for normative information about the learning task or for communication within a many-person game or simulation monitored by computer.

B. Processing Capability, Handling Character as well as Numeric Information

C. Adaptability to Specific Tasks, i.e. Convenience for Describing Models, Drawing Diagrams or Retrieving Information

D. Generality of Procedures, e.g. Separation of Procedure from Content, and Generation of Material from General Rules

Instructional programs in which the content is described separately from scoring and control procedure are easier to prepare and modify than those in which all functions are combined in one set of statements. The content can be altered or replaced without changing the algorithms and conversely, and relative effectiveness can be studied as a function of the setting of control parameters, etc.

For some learning exercises, the writing of such a rule to generate a large number of variations will prove more efficient and accurate; a larger number of items may be described more quickly than if the author were forced to write them all out, and it reduces the probability of oversight or error on the part of the author. At other times, however, when the number of examples needed is fairly small or the rule is difficult to compose, the author can save time by writing out each needed variation.

E. Manipulation of Files, i.e. Directories, Curriculum Material, Performance Data, etc.

The facility for recording information in a log is of special significance in education applications of computers, and was not typical of earlier systems not prepared specifically for computer instruction. Furthermore, the record of program status and of the occurrence of particular transactions are needed for on-line decisions.

The best tactic for record-keeping in a research-oriented system may be to write continually a log of everything which happens, and then let the researchers pick out what they need later. However, an operational system servicing students and teachers economically should log only the information certain to be needed and in a format suitable for quick and inexpensive summarisation for use by learners and managers of the instruction.

Files are very important in a time-sharing system, and even more so in the instructional milieu. Generally a hierarchy of files should be available, the heavily-used files on disk storage and larger or backup files on tape or data cell since these modes are cheaper. Temporary files are necessary so that the terminal user can do "scratch" calculations. Some scheme of access should allow various read, write, read-only and write-only privileges to users, in accordance with the user's status.

3. GENERAL CONSIDERATIONS OF UNIVERSALITY

A. Universal Language by Established Standard

It is urgent that serious consideration be given to means for translating instructional materials and strategies from one institution and system to another. A standard or universal programming language is assumed to be the key.

Many people complain about proliferation of programming languages for instructional uses of computers, but few people are willing to let anyone else do something about it. Each project, each manufacturer, almost each individual user establishes preferences, working habits, etc. and would not like anything like standards imposed on him by someone else.

Attempts to impose a single major standard language (or a small number of languages) almost certainly will fail to establish translatability among institutions. Even if one could assemble enough support to produce a definition of a standard language, imposing this standard would be nearly impossible. Although the allocation of federal funds for curriculum development might be made conditional on that standard, funds will continue to come from a variety of sources, including the individual institutions who generate material for their own use.

Although strong forces will be encountered against standardisation, one common language is not the important goal. Because of the great variety of purpose and process in instructional programming, a common language is less desirable than it might be in business or scientific programming.

If there is to be only one language which all users must share, then it must be some notation or set of conventions for describing computer-based learning exercises, or more generally, uses of computers and information processing in support of learning and instruction.

B. A Few Common Languages as Justified by Different Requirements

Different purposes require different languages. In the exploratory phases the author (or research team) should have two or more procedure-oriented languages available, e.g. FORTRAN and SNOBOL, or PL/I and LISP 1.5. During latter stages of curriculum development, and in actual use with learners, the authors should have suitable procedures worked out and compiled (or coded in assembly language) for efficient operation, e.g. three alternate drill strategies, two modes for explanation and expositions, and a number of task-oriented environments. Perhaps one of the languages for exposition would look like COURSEWRITER or PLANIT, but it is not necessary and perhaps not desirable to begin there.

Achievement of standards with a different language for each identifiably different task is probably less likely than effecting a single language standard. All the problems of achieving agreement, acceptance, and widespread use are multiplied. However, the essence of "universality" is not standardisation but translatability.

C. Automatic and "Manual" Translation Among Languages of Similar Purpose

New languages and systems will have greater capacity for translation of instruction programs from present programming languages in which they were implemented. Translatability is possible without imposing any restrictions on innovative ideas for language or strategy.

Investment in automatic translation from one language to another is an appealing concept; differences among learning exercises in regard to procedural aspects are disappearing. The major problem is the considerable cost of writing these translators, and maintaining them as various languages are changed.

In some cases automatic translation is not possible because of essential differences in hardware. One system may lack essential clock or interrupt features. Functional differences occur in the input and output facilities, that is, the equipment used to display information to the learner and accept his responses.

The materials and strategy for one course were transferred to a new computer system by writing programs which automatically generated a set of new instructions for the second machine. The original course designers had conceived of the package of lessons in a general way so that the number of generators that had to be programmed was relatively small. Success with this approach to translation in part depends on the extent to which the curriculum designers separate data from procedure, i.e. content from strategy.

The current trend in translator writing systems (compiler-compilers, macro-generators, etc.) may provide for diversity within a common environment. The general functions of information processing, data structures, etc. are provided in a basic system. Each group of users still could extend and adapt the capabilities of the system to its particular task and for its convenience. The elementary functions or processes would remain a common standard, and translation could be made through an experienced programmer who reproduces the capabilities rather than the course.

Standardisation, or more reasonably, "translatability" of computer-based learning exercises from one system and project to another requires attention to hardware as well as software. Important aspects of curriculum developed for a system rich in interactive capability (audio,graphic, etc.) may have to be dropped when moved to another system limited to typewriter input and output.

Clearly some useful work has been done with limited terminal capability. If one begins with the idea of adaptability to various computing systems and terminal devices (e.g. to most general-purpose time-sharing systems available in schools of engineering across the country), the problem appears solvable. A specific instance in the engineering area is the distribution of applications packages (STRESS, COGO, etc.).

Perhaps the computer-based learning exercises which are a) most translatable, and b) most worthy of translation, are those which are viewed by prospective users as tools or open-ended exercises. A tool which an instructor can provide his students in situations of his choice and with his best advice will have a much broader audience than a programmed instruction exercise which decides all contextual considerations for the instructor. Just such a tool is more readily translated to other computers and programming systems than the CAI materials with a closed approach.

Discussion of standardisation and translatability is confused by failure to distinguish among different kinds of users and different levels of documentation. Automatic translation requires complete knowledge of two systems, and is extremely difficult or impossible if the intention to translate between two systems was not considered in the design of at least one of them. Manual translation by an experienced programmer requires documentation of one type; adoption by another user requires "documentation" of another type. Even with automatic translation of the basic code, the learning exercises may remain unused if the instructor/manager in charge has no convenient way to assess the content and methods of the exercise.

D. Communication and Documentation with a "Publication" Language

Documentation has two main functions, that of information transmission and work simplification. It transmits information to potential users concerning: (1) contents of instruction and (2) effective use and application of the program. It simplifies work by: (1) enabling the user to find actual or potential trouble spots; (2) assisting the user to eliminate problems which may arise, and (3) simplifying revision.

At the same time one considers the means and costs of various kinds of translations of computer programs (whether automatic, manual, or a mixture), one must also consider means for informing the individual (professor, administrator, or even individual student) who must first decide whether to spend the resources to accomplish the translation.

1. Among curriculum developers

Representation of a procedure statement for a curriculum expert not accustomed to computers requires an approach different from the standardised flow charting used by computer specialists.

Most languages are not suitable for describing the content and strategy of a learning exercise, and other means for documentation are rarely used by the authors. A significant portion of a two-million dollar budget for curriculum development and operations can be absorbed by additional staff effort necessary to program interesting strategies with a language which is not suitable. Typically nothing is left for documentation and distribution.

The separation of content (definition, facts, relations, etc.) from procedure (rules for review, error checking, etc.) makes documentation and translation a much easier job.

A communication medium for talking about instruction will promote design of more reasonable learning tasks, and serve also as a significant tool for advancing instruction research and strategies of curriculum development.

2. To reviewers and potential users

The difficult task of selecting a textbook or reference source for students is complicated when the author hides part of his material in a computer (along with some strategy for gradually revealing it to students). A potential user should not have to unscramble the cryptic computer program listing, or extract pieces paragraph-by-paragraph at a teletypewriter or CRT.

In most cases the essential information about a computer-based learning exercise can be derived without executing the program; careful study of proper documentation should provide all information a potential user needs about the materials and logic.

Relevant information is obtained more efficiently through organised exploration of a description of the program than through reading individual records of student-machine interaction or through blind searching on-line at a student station for the eventualities for which the author has provided coding.

Actual experience with a computer-delivered exercise may be an important factor in understanding and evaluating an instructional unit, especially if certain knowledge and technique are supposed to unfold or develop during the learning experience. An important component of some learning experiences is affective, that is, success depends on an impression or feeling of pleasure, satisfaction or possibly surprise. Negative experiences might also be identified by a curriculum reviewer in on-line experience more readily than in an author's statement of specifications.

3. To programmers

Within a number of applied research projects some means has been developed for curriculum designers to communicate with computer programmers: tables, problem formats, special notations, etc. These temporary measures have shaped the continuing evolution of programming languages for instructional systems, and could be formalised into a suitable "publication" language.

Humans can interpret by context many statements which automatic language processors find ambiguous, and this machine deficiency can be corrected only at considerable expense of programming and processing time, if at all. On the other hand, the computer programmer implementing a lesson should receive his instructions from the curriculum designer some relatively constant notation which can be interpreted quickly and accurately.

Designers of computer-based learning exercises should be able to communicate directly with the computer. Whenever an intermediate programmer has to be called in, he should respond in a way which not only meets the immediate need but provides automatic (computer) handling of future requests, i.e. direct instructions from the subject expert to the machine.

E. Natural Versus Formal Language

The designer of a learning exercise should be able to instruct the computer system in a language natural to him and to his discipline, unconstrained by artificialities of computer notation and operation.

The formality of (most) computer languages is a good thing, requiring of the user increased attention to relevant details of his procedure. If one were able to speak to computers in completely unconstrained English, an impossible situation at least for a very long time, his directions would almost certainly lack the specificity required for determination of automatic prescriptive assistance for self-instruction.

Formal language is desirable in instructional technology for a number of reasons, among them: tacit assumptions are excluded; ambiguity is reduced; description of procedure becomes more readable; and generalised procedures can be applied in other situations. In general, a formal language appropriately requires the user to reflect on what he instructs the machine to do.

Education and training in a discipline, in particular, the practical application of techniques to the presentation of self-instruction and other individualised learning materials, suffers because of inadequate communication with the English language. The community of users must use a relatively unambiguous language for discourse about purposes and procedures before it will benefit from a language for computer implementation.

4. SYSTEM LIMITATIONS ON LANGUAGE

A. Hardware

Uses of auxiliary memory for updating formatted files of student records and making decisions in real time on the basis of certain aspects of the data require direct access to specific portions of the information. It is disappointing to find the disk and drum storage on conversational computing systems used in a tape-like fashion instead of as the direct-access file devices they really are.

When special symbols are required as in language, mathematics and the sciences, a printer-type terminal device will need special printing elements (as in the IBM selectric type ball), or an electronic display will need facility for user-defined characters to be generated (as on a CRT or plasma discharge panel with appropriate hardware attachments).

B. Software

When the subject expert and educational technologist become distracted from their real purposes by the peculiarities of current computer systems and programming languages, work should leave the computer for a time until the essential parameters of the learning situation are determined. If specifications for human tutoring are prepared as if for a more sophisticated computer system than now available, techniques developed off the computer will more readily be adapted for computer implementation later.

A broadly conceived instruction system probably should begin with a general-purpose system and add facility for moving from the tutorial mode into other user sub-systems and returning when an exercise is completed. The author of a problem set may need to maintain contact with the student through some means of monitoring his work on a problem, and then bring him back to the tutorial mode because of elapsed time, number of problem attempts, or even an anticipated error which requires special attention.

Instructional systems should incorporate many programming capabilities which can be used by both author and student. In addition to simple computational aids, some lesson designers will want to provide an algebraic language, a text-processing language, a model-building or simulation language, perhaps a specific system or model written for student use, or information organisation and retrieval capability.

C. Communications

Each terminal device for communication between the computer and the user has physical and logical characteristics which determine the kinds of instructional techniques and/or computer system configurations for which it may be suitable. The suitability factos include facility for messages from users to computer (input); messages from computer to user (output); distance between computer and user; cost of communication link; cost and reliability of device.

The rate of message transmission from student to computer ranges widely for different applications, but it averages out to about one keypress every two seconds, including the time for reading and thinking. The machine sends messages to each user in a burst but the rate averages out to about two characters per second.

Communication costs usually are significant in servicing remote terminals in large numbers and/or at long distances. Since some devices require a voice-grade telephone channel but leave it 99% unused, some arrangement for multiplexing will allow up to 100 terminals to be serviced by a single line between the computer and the site of the cluster of terminals.

If a diagram or picture must be read from a video file associated with the central computer, much communication capacity will be required to get it out to the local terminal quickly. Alternatively, it can be sent slowly before it is needed, and then displayed as often as necessary from local storage associated with the terminal.

D. Summary of Cost Considerations

Other media than computers will continue to be less expensive for storage, presentation and testing; the economics of computer use are more favourable for practice and recitation exercises, where a greater degree of exchange between learner and data base is typical.

The response time and operating costs for any particular user depend in large measure on the priorities established for that kind of user when the system was designed and tuned.

Techniques for preparing curriculum files must be more powerful in the sense of fewer hours required of the subject expert to write and revise materials which achieve the objectives intended of the learning experience. Authors cannot often afford the luxury of individually shaping or tailoring each line of text in each frame for each kind of student.

It is today cheaper, and in some instances perhaps more convenient, to handle some desirable translator features manually with clerks and writing assistants. The next important step is careful development and evaluation of language features which adapt to the needs of authors and subject areas.

Conversational languages emphasise convenience, and sometimes require considerable additional cost in computer time during execution. The number of operations for interpretation of a symbolic program is always greater than for execution of a program already compiled into machine-level statements. Of course a user may be willing to pay more for execution if his results will be available immediately and without complication, along with quick diagnostics and opportunities for changes in the program at stopping points throughout.

5. DESIGN CONSIDERATIONS

Recent adventures in time-sharing warn of the inherent difficulties in such endeavours. Initial hardware investment is heavy; staff members must be very competent and well paid; results lag far behind effort invested in the project. A point too often overlooked by planners of new instructional projects is that time-sharing systems need large development resources, much larger than most researchers can afford.

A. Adaptability

Facility for definition of functions should be extended to provide for definition of a) character operations as well as numeric ones, and b) distributed operators which apply throughout one or more statement lines. The latter would allow for definition of new operations with convenient formats for specifying answer processing. More than one line should be permitted in the definitions, and the possibility of an operator being distributed among two or more variable names must be allowed in the parser.

One way to extend a language to handle additional applications is to provide linkage to other programs. No one language now available can handle the variety of applications efficiently, and some useful subroutines may already be available in other languages on the same system. The major problems seem to be: 1) transferring data, 2) returning control to the calling program, and 3) leaving the user in control in spite of program or system errors.

The problems with extending a language through definition of new operators and statement types concern the internal representation of the language, simple rules for describing new features, and the ability to recognise operators distributed throughout a list of variables even on more than one line or program statement.

It is not obvious what the elements of programming should be. The basic statements and operations need be elementary enough to permit building the variety of processes desired by programmers. However, high level commands should be assigned to frequently used routines constructed by programmers in a way that the syntax can be readily used by curriculum designers.

B. Economics

Variety and flexibility in programming capability of an instruction system are not necessarily incompatible with economical operations. Early decisions by system designers about specifically what is needed by users inappropriately limit the scope of applications.

New features defined within an interpretive language for execution as needed must be reinterpreted each time the function is used, and little economy of execution results. The ability to compile or assemble a routine, link it to the interpreter, and specify its execution in a statement form natural to the user will increase convenience while making certain information processing operations more economical to perform.

One way to accomplish some economic advantage is to reassemble the interpreter, adding the new statements, functions or operators to the language. This delays availability unless an informed system programmer is always at hand. Reassembly for one user also raises some questions of proliferation: Should he then have his own special version; do changes in the basic compiler take effect for everyone?

A recent addition to the tool kit of a system architect is microprogramming. The machine's instruction repertoire need not be wired-in; rather the processor is itself an interpreter of microprograms which are loaded in special memory, one for each instruction. For interactive and conversational uses the savings can be substantial in both time and speed.

Because background jobs have no response-time constraints, they are ideal for using any excess (idle) processor time. However, such jobs could destroy any benefit by slowing interactive responses and forcing greater overhead. Fixed memory can be allocated to the resident background jobs, but fine tuning is the tricky part.

C. Modularity

Language processors are usually designed in modules. Logical separation facilitates locating an error in the processor, introducing changes, and reprogramming the processor for use within another operating system.

It is not the modular concept but sensible programming which makes a difference. Separation into blocks of statements which have little if any interaction is only a way to encourage sensible programming.

D. Documentation

Encouraging uses of a system and language which has inadequate documentation is likely to lead to disappointment for users and frustration for those responsible for maintaining service. Errors or other considerations requiring modification are certain to arise, and the processors should be adequately described for maintenance purposes.

6. INTERACTIVE MODE CONTRIBUTIONS TO LEARNER AND AUTHOR

A. Immediate and Responsive Reply

The essential contribution of interactive programming must involve responsiveness of the system, and this factor provides special benefits for the casual and infrequent user. He may be well advised, when unsure of the proper syntax, to try various likely ways until the interpreter accepts one and does what he intended. Better yet, the processor should tell him what form to use the first time an uninterpretable statement is entered, or refer him to the section of a reference manual which is likely to explain away his confusion.

If diagnostics, provided at the moment and backed up by references to readily available literature, can relieve the user of concern for the means to describe his procedure, he will give more attention to solving the problem. A shorter elapsed time between problem definition and solution, and the time savings attributable to continuous working sessions provide another bonus.

Much of the enthusiasm for conversational computing languages may relate to non-essential features; quick response and understandable diagnostics can be provided in batch systems.

B. Ease of Conducting a Dialogue and Learning the Rules

Interactive programming languages incorporate aids for program testing in a very natural way. The same statements with which stored programs are written can be used as direct commands to the computer to print the values of selected variables, assign new values to test other parts of the procedure, and resume execution with any line or segment of the program.

A rather deep search for the locus of a syntax error and some attempt to interpret the intention of the user in spite of ambiguity should help along the dialogue between user and machine. This requires a cleverly written processor with auxiliary memory and decision rules which generate special user assistance.

Naturalness is an important factor in using a language, and is achieved by internal consistency as much as by relation to native language. General conventions should apply throughout; the user should be able to predict a rule he hasn't been told yet, and one aspect of the notation should not interfere with his recollection of another.

The dialogue between user and program should be truly a dialogue. That is, there may be time when the computer should take the initiative, setting up stylised instructions and asking leading questions, and other times when the user takes over. However, throughout this exchange, each may interrupt the other to suggest a new arrangement.

C. Flexibility During the Working Session

Interactive mode of work should provide opportunity for sketching out an idea, testing parts of it, going back to fill in detail and make corrections, etc. The user should elect an on-line environment because it helps him conceptualise a procedure and solve a problem, not simply because it is an available way to enter a program into a computer.

Somehow a processor might recognise when a user is making temporary notes and when he wishes his work to be saved for future use. At least the user should be given a convenient notation for designating the expected permanence of current instructions, and a means to retrieve later something found to be of greater value than originally perceived.

V. REFERENCES WITH ANNOTATIONS

Adams, E.N., "Reflections on the Design of a CAI Operating System", AFIPS Conference Proceedings, Vol. 30, 1967 Spring Joint Computer Conference, Thompson Books, Washington, D.C., (SJFF), pp.419-24.

The problems which Adams had to work around in the IBM 7010 experimental Coursewriter are typical for CAI systems which used available equipment. Many technical problems may be solved for instructional users by following developments in general-purpose systems.

The Core-partitioning and disk-fetch problems discussed are alleviated in a virtual-address environment. Use of drum and relocation hardware are more suitable than the swapping scheme mentioned.

He provides a useful analysis within his context.

Bitzer, D., and Skapendas, "The Design of an Economically Viable Large-scale Computer-based Education System", University of Illinois Computer-based Ed. Research Lab., CERL Report No.X-5, also a Report to the Commission on Instructional Technology, 1969.

The authors argue for economic viability of an instructional system. The estimate of ten man-years for the system development is reasonable, and the "plasma" terminal upon which all dreams rest is rapidly approaching commercial status.

Other aspects in the cost estimates raise questions. For instance, will authors adjust to writing computer instructional material? Scholarly books and some texts provide prestige and profit, and usually fulfil the "publish or perish" dictum. Although some time sharing services do install applications programs and pay the author a use-rental, and at least one professional journal is reviewing computer-based learning exercises, the area of materials and authorship is a weak component of the Illinois plan applied elsewhere.

Adams, E.N., "Technical Considerations in the Design of a CAI System", in the Proceedings of an NCET Seminar on Computers in Education held at Leeds, England, September, 1969.

A good tutorial for persons interested in instructional use of computers who are not well informed about hardware and system considerations, although a number of specialised terms are left undefined. The document would be a useful guide also if references were included for more detailed information about terminals and time sharing systems.

The author provides a useful organisation and conceptualisation of technical considerations (for those who already know the content), e.g. terminal and communication options; separation of procedure and content; isolation of control functions; generative techniques; and simulation.

The discussion of systems is weak, perhaps because software considerations are only half based in computer science and half in education. More work is needed to relate data about hardware, point by point, to the administrative, psychological and instructional considerations.

Breed, L.M., and Lathwell, R.H., "The Implementation of APL/360", in Melvin Klerer, Juris Reinfelds (eds), Interactive Systems for Experimental Applied Mathematics, Academic Press, New York, 1968, pp. 390-399.

This implementation was done with both the APL characteristics and the idea of a dedicated system in mind. The result has:

a) A supervisor which allocates all system resources according to a single comprehensive strategy;
b) A system design influenced by an advance analysis of APL user programs (hopefully representative);
c) Reduced overhead in the language interpreter because of attention to detail (via specialised programming techniques).

APL is easily interpreted in source form. Thus there is no translation to a syntactically rearranged internal form; only a lexical replacement is done on the input. Run-time analysis uses transition state diagrams, an efficient top-to-bottom method.

A number of storage management and swapping tips are included, as well as short discussions on error recovery and system self-monitoring. Indirectly it supports the advice that system building is not for novices (such as CAI project directors).

Engvold, K.J., and Hughes, J.L., "A Multi-function Display System for Processing and Teaching", IFIP Congress 69, Amsterdam, North Holland.

A trimodal instructional system (author, student, or program) is proposed. The article describes an implementation of the arrangement for an IBM 360 Model 50 driving an IBM 2250 graphics terminal. Engvold's article is very similar to an earlier one (CACM, Vol.10, No.16, p.339) describing a similar system using an IBM 7040. The 2250 is flexible and fun, but rather expensive and really more powerful than necessary. What us is 8K of independent storage unless the student is doing very, very complex engineering graphics? The display of Bitzer would suffice for any of the authors' demonstrations, and for most of what they propose.

Frye, Charles H., "CAI Languages: Capabilities and Applications", Datamation (September, 1968), Vol.14, No.9, pp.34-37. Reprinted in Richard C. Atkinson and H.A. Wilson (eds) Computer Assisted Instruction: A Book of Readings, Academic Press, 1969.

Frye classifies languages used for instruction as: 1) conventional compiler languages, 2) modified conventional languages, 3) interactive languages, and 4) special instructional author-languages.

Using these language categories the author considers: 1) user orientation, 2) lesson handling, 3) record handling, 4) conditional branching, 5) answer matching service routines, 6) calculation features, and 7) communication devices.

Some of the information is misleading and the general discussion is not supported by specifics but is nevertheless useful. Consideration of general-purpose along with special-purpose instructional languages is important. More could be made about some truly essential or primitive features found in many CAI languages.

Frye suggests that instructional author-languages arose because standard programming languages were too difficult to use and experienced programming help was too expensive. He should also recognise that although the prospective author need not learn as much to use a specific instructional language, he will not find it easy to deviate much from those programming styles and learning tasks which originally inspired the language.

Furthermore, new techniques for extending general-purpose languages (or generating special adaptations) will make relevant computer capabilities more accessible to non-specialists, among them the designer of computer-based lessons.

Glass, R.L., "An Elementary Discussion of Compiler/Interpreter Writing", Computing Surveys, (March 1969), Vol.1, No.1, pp.55-77.

Elementary techniques are discussed for the translation of programming languages. Detailed discussion centers around a PL/1 interpreter which the author uses for an example.

A good annotated bibliography follows the article.

Johnson, B.F., "Design of an Operating System for the Control of Student Terminals in a Computer-based Instruction System, "IFIP Congress 1968, Amsterdam, North Holland.

The RCA project described is quite straightforward technically. The use of re-entrant teaching programs is a good feature, and quite common for implementation of system programs; the teaching strategies are(included)as system programs in the RCA system.

A background job stream should use residue CPU time and various other modes of computer use should be encouraged by additional facility. The system is rather narrowly conceived but well executed. It did have to be redone to get it running in the NYC schools the next year; current documentation is available from RCA, and user opinion from schools in NYC and Pontiac, Michigan.

Lyon, Gordon and Zinn, Karl L., "Some procedural language elements useful in an instructional environment", Working paper for Project CLUE, Center for Research on Learning and Teaching, Ann Arbor, Michigan, 48104.

This paper attempts to attack the instructional language problem on the following frongs: a) given reasonable demands, what programming language primitives adequately meet the requirements?; b) how are the language features (or primitives) reflected in contemporary languages?

Suitable language primitives are discussed as parts of some general-purpose language. If the language will be used in interactive mode, some parts must be implemented as an interpreter to allow very late binding times needed for flexibility.

If viewed as an attempt to cast light on procedural features in instructional computing, the paper may succeed. However, the hypothetical language should not be taken very seriously. It is cumbersome, vague, and patch-work: at best a sketch, certainly not a blueprint.

Morton, M.S.S., and Zannetos, Z.S., "Efforts Toward an Associative Learning Instructional System", IFIP Congress 1968, Amsterdam, North Holland.

The authors propose a computer-driven interactive terminal linked to an associative memory with flexible search procedures to solve: a) lack of integrated instruction material; b) the inflexible pacing and sequencing of students. Postulated characteristics of the system include semantic content association and learning (via pattern matching and adaptive characteristics).

The semantic memory has been programmed in much simplified form. An accounting course provided material formal enough so that keyword searching provided adequate "pattern matching". Pointers in the associative structures linked to other key words, thus providing rudimentary inferential powers.

The authors propose that semantic content, pattern recognition, adaptability (and, in addition, hardware flexibility) are the basic components of instructional teaching. An exhaustive testing of the prototype may provide some indication, perhaps encouraging an implementation which comes closer to testing the authors' hypothesis.

Tonge, F.M., "Design of a Programming Language and System for Computer Assisted Learning", IFIP Congress 1968, Amsterdam, North Holland.

This is a well balanced summary of a system which became only partially operable at the University of California at Irvine. The view of early CAI languages as "five years behind the state of the programming art" rings true. The section on the requirements of the UC Irvine system is edifying, and some postscript on continuing problems would be useful.

Tonik, Albert B., "Development of Executive Routines, Both Hardware and Software", AFIPS, Fall Joint Computer Conference, Vol. 31, pp.395-408, Thompson Books, Washington, D.C.

This tutorial expands from an elementary executive to a rather complex one. Topics include: simple executive, multiprogramming, paging, and multiprocessors.

In addition to the main article, there is a chronologically ordered bibliography with articles dating from 1948.

Zinn, Karl L., "A comparative study of languages for programming interactive use of computers in instruction", Final report under ONR contract N00014-68-C-0256 (February 1969), EDUCOM, 100 Charles River Plaza, Boston, Mass., 02114, 1969a.

This report was preceded by the author's two entries in this bibliography which are more available. It includes much of the detail promised in the other two: examples of actual code, summaries of languages, a discussion on interactive languages, a glossary of terms, aspects for a system taxonomy, and documentation guidelines.

More interpretable and directly useful work is in progress.

Zinn, Karl L., "Programming conversational use of computers for instruction", Proceedings of the 1968 ACM National Conference, Brandon/Systems Press, Inc., Princeton, N.J., 08540, 1968.

From the 30 or so available languages for conversational instruction only 3 or 4 really different kinds have appeared. Author suggests: 1) successive frame, 2) limited-context conversation 3) presentation of a curriculum file by a standard procedure, and 4) data analysis and file editing.

Four types of users are considered: instructors, authors of instructional strategies, instructional researchers, and programmers and systems people. Further discussion explores languages in a general-purpose time-sharing environment. Extendability is mentioned. The article concludes by comparing two low-cost extended languages (FOIL and FORFIT) with two standard languages for conversational instruction (COURSEWRITER and PLANIT).

Zinn, Karl L., "Languages for programming conversational use of computers in instruction", IFIP Congress 1968, Amsterdam, North Holland.

An outline of a comparative study of existing languages (then in progress) with tentative suggestions for improvements. Recommendations are derived from comments by authors of materials for various systems. The author also describes languages under development for a general-purpose system at the University of Michigan (IBM 360/67 using the Michigan Terminal System).

Zinn, Karl L., "Implications of programming languages for instructional uses of computers in mathematics", in CAI in Mathematics Education edited by Ralph T. Heimer, Washington, D.C.: National Council of Teachers of Mathematics, 1969b.

A tutorial presentation on instructional uses and programming languages with examples taken from computer-based curriculum prepared for mathematics education. The section on kinds of instructional programming has been rewritten for Project CLUE and a meeting on computers in education sponsored by OECD-CERI.

Zinn, Karl L., "Instructional programming languages; A five-year perspective", Educational Technology, in press for March, 1970.

Transcript of a presentation made at AERA in February of 1969. Criticises present languages and techniques for frame-oriented instructional programming and suggests new approaches to achieve effective and economical instruction by computer in the tutorial mode. Recommends problem solving and procedure-writing uses for now because of economy and greater accessibility by individuals and smaller institutions. Some of the conclusions have been incorporated in the background statement prepared for the OECD-CERI meeting.

CURRENT SOURCES:

ACM Special Interest Group on computers in education, a subgroup of the Association for Computing Machinery, plans presentations and discussions for national (USA) and international meetings; publishes a Bulletin of news, abstracts and technical notes available from ACM Headquarters, 1133 Avenue of the Americas, New York, New York 10036; and maintains panels for review of matters of interest to the profession such as bibliographies, abstracting services, comparative studies, and coverage in journals and other publications.

The Commission on Education of the National Academy of Engineering maintains a Committee on Instructional Technology which can be expected to give considerable attention to computer uses in education in the near future. The report of a previous study and information about new studies in progress can be obtained from David Miller, National Academy of Engineering Commission on Education - JH611, 2101 Constitution Avenue N.W., Washington, D.C. 20418.

The Michigan Education Research and Information Triad, established to promote instructional uses of computers in institutions of higher learning in the State, maintains a file of documentation on languages used for instructional programming with samples of use. Some continuation of the Educom comparative study of programming languages is likely; interested persons can check with the Associate Director for U-M, MERIT, 611 Church Street, Ann Arbor, Michigan 48104.

VI. GLOSSARY OF SELECTED TERMINOLOGY FOR COMPUTER USES IN EDUCATION

- access time. Time required to obtain information from storage (read-time) or to put information away in storage (write-time).
- acoustic coupler. A device used in place of a data-set to transfer information from the terminal via an ordinary telephone over telephone lines to the computer and vice versa.
- AHI. 'Augmentation of human intellect'. Computer techniques for retrieving, re-arranging and manipulating information, usually text, sometimes diagrams, or anything that helps one engage in intellectual activity; computer extension of human abilities to accomplish instruction research, composition or other creative work.
- algorithm. A procedure for solving a problem. When properly applied, an algorithm always produces a solution to the problem. (Compare with "heuristic".)
- analogue computer. Device using voltages, forces, fluid volume or other continuously variable physical quantities to represent numbers in calculations. It is convenient for solving differential equations, simultaneous equations and equilibrium problems. (See "digital computer".)
- ASCII. American Standard Code for Information Interchange. Established by the American Standards Association as the standard for representation of numbers in computing machinery.
- Bandwidth. The difference, expressed in cycles per second, between the highest and lowest frequencies of a band or part of a channel; a determinant of amount and quality of information which can be passed per second. Bandwidth is measured in cycles or bits per second (cps or bps), kilocycles per second (KC) or megacycles per second (MC).
- Batch processing. A method of operation in which a number of similar jobs are accumulated and processed together, usually being done in serial order. (See "time-sharing" for contrast.)
- binary device. Having two states; on-off, yes-no, true-false.
- bit. (contraction of "binary digit"). A unit of information content; the smallest element of binary computer memory or logic.
- branching. Altering the course of a set of instructions by switching when some predesignated event occurs.
- buffer. A storage device used to compensate for difference in rate of flow of data, or time of occurrence of events, when transmitting data from one device to another.
- byte. A group of bits (usually six to eight) representing a character, for some computers the smallest addressable unit of memory.

- CAE. Computer-assisted education; computer-augmented education.
- CAI. Computer-assisted instruction; Computer-aided instruction; Computer-augmented instruction. Defined narrowly, it refers to tutorial exercises or computerised programmed instruction; broadly defined, it encompasses the entire field of computer uses for instruction in which there is an interaction between student and machine (e.g. drill, tutorial, simulation, problem solving, and scholarly aids).
- CAL. Computer-assisted learning; also the name of an on-line computation language developed at Berkeley, and a coursewriting language developed at Irvine.
- CBI. Computer-based instruction. Similar to CAI and CAL but less used.
- CBL. Computer-based learning.
- channel. A path for electrical transmission between two or more points. Also called a circuit, facility, line, link or path.
- character. A digit, letter or other symbol, usually requiring six or eight bits for representation in digital computers.
- CMI. Computer-managed instruction. The main function of the computer in this case is to assist the teacher in planning instructional sequences. The actual instruction may or may not involve the computer.
- compiler. Computer program for translation of instructions expressed in a user language (e.g. albegraic formulas, logical expressions or transfers of control) into a machine language (e.g. binary numbers signifying basic operations such as add, compare, store and jump).
- core. The rapid access memory of a central processing unit; usually made of many small rings (cores) of magnetic material which may be in either of two states of polarisation.
- course. Used rather loosely to mean any instructional sequence or computer-based learning exercise.
- CPU. Central Processing Unit. The central section of a computer including control, arithmetic and memory units.
- CRT. Cathode ray tube. In common use as a television-like display device for drawings and text.
- cursor. A point or line of light displayed on the CRT and under the control of either the user or the computer to indicate the point at which the next display or editing operation is to occur.

- data-phone. A trade mark for the data sets manufactured and supplied by the Bell System; a service mark for the transmission of data over the regular telephone network (DATA-PHONE Service).
- data-set. A device for transmission of data over the regular telephone network.
- debug. To search for and correct errors ("bug") in a computer program.
- diskpack (also disc). A stack of disk-like plates coated with magnetic material for the storage of information; bits can be stored upon and read from surface while pack revolves at high speeds, somewhat like a stack of phonograph records crossed with a tape recorder.
- down-time. Time when a computer is not available for operation, usually because of a failure in the equipment.
- drum. A cylindrical drum coated with magnetic material for the storage of information. Bits can be stored upon and read from surface while it revolves at high speeds.
- duplex. In communications, pertaining to a simultaneous two-way and independent transmission in both directions (sometimes referred to as "full duplex"). (Contrast with "half-duplex").
- facsimile (FAX). Transmission of pictures, maps, diagrams, etc. The igage is scanned at the transmitter, reconstructed at the receiving station and duplicated on some form of paper.
- feedback. In programmed instruction, providing the student with information on correctness of his last output or response. The feedback may be designed to correct a student's incorrect response.
- flag. An error return found in the right margin of the compile listing.
- flow diagram. A schematic or block representation of programming strategy.
- frame. The smallest unit of programmed instruction usually consists of information and/or a question, an opportunity for an answer, and some provision for checking the answer.
- generative techniques. Standard patterns or procedures, or algorithms applied to curriculum files for the generation or assembly of sequences of instructional materials; an alternative to frame-by-frame programming.
- half-duplex. Pertaining to an alternate, one-way-at-a-time, independent transmission (sometimes referred to as "single"). (Contrast with "duplex".)

- hardware. The equipment components of a computer system; the machinery as opposed to the programs which are run on the machinery.
- heuristic. A guide to finding a solution to a problem that cannot be proved to always result in a solution.
- instructional programming language. A computer language or notation particularly suited to the description of instructional procedures for computer delivery. (See "programming").
- interactive. Computer operation providing for exchange between user and program (or system), whichever may take the initiative. An interactive drill program checks the time and accuracy of answers and responds immediately to the user; an interactive problem solving system responds to solution attempts by the user and allows modification in the procedure at the moment. (For non-interactive, see "batch").
- interface. A shared boundary, for example, the boundary between two sub-systems or two devices.
- interrupt. A hardware feature which allows the computer to stop working momentarily on one task, handle the interrupting task, and return to the first without losing information or interim results of processing.
- INWATS. Similar to WATS but allows inward calls on a flat monthly rate.
- I/O. Input/output of information to and from computers; usually refers to devices such as an electric typewriter, card reader and punch, paper tape reader and punch, printer, etc.
- IPI. Individually prescribed instruction. Originated with an instructional project at the University of Pittsburgh's Learning Research and Development center, now used widely as label for strategy of individualising selection of exercises and rate of work for each student.
- K. Thousand; e.g. 32K words of memory means 32,000 words of computer memory.
- LDX. Long Distance Xerography. A name used by the Xerox Corporation to identify its high speed facsimile system. The system uses Xerox terminal equipment and a wide band data communication channel.
- latency (student response time). The time from the display of an instructional stimulus to the start or completion of the student's response.
- light-pen. A photo-sensitive device used for communication with a computer via a cathode ray tube; an electronic pointer.

- linear programming. (a) Mathematical: techniques for optimising a linear function of several variables subject to linear inequality constraints on some or all of the variables; (b) instruction: the simplest form of programmed instruction or CAI; all students follow the same sequence.
- line-switching. The switching technique of temporarily connecting two lines together so that the stations directly exchange information.
- link. See "channel".
- log. To record student-computer interactions.
- memory. The storage components of a computer's central processing unit in which bits of information are stored and from which they may later be recalled. (See also "storage").
- microwave.All electromagnetic waves in the radio frequency spectrum above 890 megacycles per second.
- model. An idealised representation that demonstrates the relationships between relevant variables. Models are used to better understand and control a real situation.
- multiplexing. The division of a transmission facility into two or more channels.
- off-line. Processes performed outside of the operation of the central processor of a computing system.
- on-line. Connected directly to the central computer, e.g. an electric typewriter in direct communication with computer processor.
- operating system. The collection of programs (software) which direct or supervise the utilisation of processing components and the execution of programs.
- partition. Running the computer so that different tasks, perhaps batch-processing and time-sharing operations, are performed simultaneously. Time-share operations can be given priority.
- polling. A centrally controlled method of calling a number of points to permit them to transmit information.
- port. The physical facility for connecting a phone line from a user terminal to the computer.
- programming. (a) Instructional: the construction and arrangement of elements of learning exercise and perhaps self-testing in a way specifically designed to promote effective and efficient learning; (b) computer: the construction and arrangement of elements of a procedure specifically designed to achieve a problem solution or demonstrate a process.

- RAND Tablet. A metal writing surface developed by RAND Corporation for input of graphic information to a computer through use of a special writing stylus.
- random access. A facility whereby information can be returned from any part of a storage device rapidly at any time.
- real-time. Performance of processing during the actual time the physical process transpires, in order that results of the computation can be used to guide the physical process.
- response time. The amount of time elapsed between generation of an inquiry at a data communications terminal and receipt of a response from the computer at that same terminal.
- RPQ. Request Price Quotation. Special equipment features which might be provided but are not included in announcements and price lists.
- software. (a) Computer: programs as contrasted with computer components (see "hardware"); (b) instruction: curriculum materials as contrasted with computer facilities or peopoe.
- station. One of the input or output points on a communications system.
- storage.The capacity of an information processing system to put aside or save for future use bits of information. (See "core" and "disk").
- storage protect. A hardware feature which prohibits one user in a shared system from using or changing information stored in memory allocated to other users.
- student station. I/O equipment designed for student use in interacting with a computer.

teaching logic. A pattern or strategy for instruction into which various topics or sets of questions and answers may be placed.

- tele-processing. A form of information handling in which a data processing system utilises telegraphic communication facilities, e.g. using a computer remotely via telephone lines.
- teletypewriter exchange service (TWX). An automatic teleprinter exchange switching service provided by the Bell System.
- telpak. A service offered by communications common carriers for the leasing of wide band channels between two or more points.
- terminal. A point at which information can enter or leave a communication network, or the I/O device used at that point.

- tie-line. A private line communication channel of the type provided by communications common carriers for linking two or more points together.

- time-sharing. A method of operation in which components of a computer facility are shared by several users for different purposes at (apparently) the same time. Although each device actually services one user at a time, the high speed and mutiple components of the facility give the outward appearance of handling many users simultaneously.

- translator. A computer program which accepts statements or instructions written in one language and produces statements in another language or perhaps direct instructions to the computer for execution. (See "compiler").

- voice grade channel. A channel suitable for transmission of speech, digital or analog data, or facsimile, generally with a frequency range of about 300 to 3000 cycles per second.

- wide area telephone service (WATS). A service provided by telephone companies which permits a customer to make calls to telephones in some geographic zone on a dial basis for a flat monthly charge.

- word. A set of bits sufficient to express one computer instruction (usually 12 to 48 bits long depending upon other characteristics of the machine). Usually the equipment is wired to transfer one word of information at a time.